first order logic, for function-free programs). The region between $T_\Sigma \uparrow$ and $T_\Sigma \downarrow$ corresponds to $IF(\Sigma)$, which comprises exactly those atoms which fail infinitely. In practice, most sensible programs have $IF(\Sigma) = \emptyset$, leading to $T_\Sigma \uparrow = T_\Sigma \downarrow$ and therefore $B(\Sigma) = SS(\Sigma) \cup FF(\Sigma)$.

1.4 First-Order Logic

Suppose you wanted to express logically the statement: 'All humans are apes.' One of two ways can be used to formalise this in propositional logic. We can use a single proposition that stands for the entire statement, or with a well-formed formula consisting of a lot of conjunctions: Human1 is an ape $\wedge$ Human2 is an ape .... Using a single proposition does not give any indication of the structure inherent in the statement (that, for example, it is a statement about two sets of objects—humans and apes—one of which is entirely contained in the other). The conjunctive expression is clearly tedious in a world with a lot of humans. Things can get worse. Consider the following argument:

Some animals are humans.
All humans are apes.
Therefore some animals are apes.

That the argument is valid is evident: yet it is beyond the power of propositional logic to establish it. If, for example, we elected to represent each of the statements with single propositions then all we would end up with is:

Statement	Formally
Some animals are humans.	P
All humans are apes.	Q
Therefore some animals are apes.	$\therefore R$

But the formal argument is clearly invalid, as it is easy to think up arguments where P, Q are *true* and R is *false*. What is needed is in fact something along the following lines:

Statement	Formally
Some animals are humans.	Some P are Q
All humans are apes.	All Q are R
Therefore some animals are apes.	$\therefore$ some P are R

Here, P, Q, and R do not stand for propositions, but for terms like *animals*, *humans* and *apes*. The use of terms like these related to each other by the expressions 'some' and 'all' will allow us to form sentences like the following:

All P are Q
No P are Q
Some P are Q
Some P are not Q

The expressions 'some' and 'all' are called *quantifiers*, which when combined with the logical connectives introduced in connection with proposition logic $(\neg, \wedge, \vee, \leftarrow)$, results in the powerful framework of first-order or *predicate logic*.

1.4.1 Syntax

The language of predicate logic introduces many new constructs that are not found in the simpler, propositional case. We will first introduce these informally.

Constants. It is conventional in predicate logic to use lowercase letters to denote proper names of objects. For example, in the sentence 'Fred is human', Fred could be represented as $fred$.

Variables. Consider the statements:

All humans are apes
Some apes are not human

Using the letter x as a variable that can stand for individual objects, these can be expressed as:

For all x, if x is human then x is an ape
For some x, x is an ape and x is not human

Quantifiers. The language of predicate logic introduces the symbol $\forall$, called the *universal quantifier*, to denote 'for all.' The symbol $\exists$, called the *existential quantifier*, is used to denote 'for some' or, more precisely, 'for at least one.' The sentences above can therefore be written as:

$\forall x$ (if x is human then x is an ape)
$\exists x$ (x is an ape and x is not human)

Predicates. In their simplest case, these are are symbols used to attribute properties to particular objects. It is conventional in logic (but ungrammatical in English) to write the subject after the predicate. Thus the sentence 'Fred is human' would be formalised as $Human(fred)$ [22]. More generally, predicate symbols can be used to represent relations between two or more objects. Thus,

[22] Logicians are a parsimonius lot: they would represent 'Fred is human' as Hf. The representation here is non-standard, but preferred for clarity.

'Fred likes bananas' can be represented as: $Likes(fred, bananas)$. The general form is therefore a predicate symbol, followed by one or more *arguments* separated by commas and enclosed by brackets. The number of arguments is sometimes called the *arity* of the predicate symbol, and the predicate symbol is often written along with its arity (for example, $Likes/2$). Formalising sentences like those above would result in quantified variables being arguments:

$$\forall x\ (\textsf{if}\ Human(x)\ \textsf{then}\ Ape(x))$$
$$\exists x\ (Ape(x)\ \textsf{and not}\ Human(x))$$

Or, using the logical connectives that we have already come across:

$$\forall x(Ape(x) \leftarrow Human(x))$$
$$\exists x(Ape(x) \wedge \neg Human(x))$$

Functions. Consider the statement: 'The father of Fred is human.' Although we have not named Fred's father, it is evident that a a unique individual is being referred to, and it possible to denote him by using a *function* symbol. One way to formalise the statement is: $Human(father(fred))$. Here, it is understood that $father(fred)$ denotes Fred's father. A function symbol is one which, when attached to one or more terms denoting objects produces an expression that denotes a single object. It is important that that the result is unique: a function symbol could not be used to represent, for example, 'parent of Fred.' As with predicates, the number of arguments of the function is sometimes called its arity.

The following points would not be evident from this informal presentation:

1. Variables need not designate different objects. Thus, in $\forall x \forall y Likes(x, y)$, x and y could refer to the same object;
2. The choice of variable names is unimportant. Thus, $\forall x \forall y Likes(x, y)$ has the same meaning as $\forall y \forall z Likes(y, z)$;
3. The same variable name, if quantified differently, need not designate the same object. Thus, in $\forall x \forall y Likes(x, y) \wedge \forall x \forall y Hates(y, x)$ the x, y in $Likes(\cdots)$ need not be same as the x, y in $Hates(\cdots)$;
4. The order of quantifiers can matter when $\forall$ and $\exists$ are mixed. Thus, $\exists x \forall y Likes(x, y)$ has a different meaning to $\forall y \exists x Likes(x, y)$. However changing the order has no effect if the quantifiers are all of the same type. Thus, $\forall x \forall y Likes(x, y)$ has the same meaning as $\forall y \forall x Likes(x, y)$;
5. "Free" variables in a formula are those that are not quantified. For example, in the formula $\forall x Likes(x, y)$, y is a free variable. In contrast, quantified variables are called "bound" variables. It may not be immediately apparent that a variable can have both free and bound occurrences in a formula. For example in $\exists x(Likes(x, y) \wedge \exists y DisLikes(y, x))$, the variable y is free in the $Likes$ and bound in $Dislikes$. x on the other hand is

bound in both (by the outermost quantifier). It is normal to call a formula with no free variables a *sentence*, and it only really makes sense to ask about the truth of sentences;

6. Negation should be treated with caution. Thus, in 'Some apes are not humans', 'not' plays the role of *complementation*, by stating that the set of apes and the set of non-humans have at least one member in common. This can be formalised as $\exists x(Ape(x) \wedge \neg Human(x))$. One the other hand, 'not' plays the role of true negation in 'It is not true that some apes are humans' formalised as $\neg\exists x(Ape(x) \wedge Human(x))$;
7. It can be tricky to match English sentences to ones that use $\forall$ and $\exists$. Thus, in 'If something has a tail then it is not an ape', the use of 'something' suggests that formalisation would involve $\exists$. The statement is, in fact, a general one about apes not having tails, and involves universal quantification: $\forall x(\neg Ape(x) \leftarrow Tail(x))$;
8. By denoting 'at least one', the existential quantifier $\exists$ includes 'exactly one' and 'all'. This does not coincide exactly with the usual English notion of 'some', which denotes more than one, but less than all.

We can now examine the formal rules for constructing well-formed formulæ in predicate logic. For the language of predicate logic, we will restrict the vocabulary to the following:

Constant symbols:	A string of one or more lowercase letters (except those denoting variables)
Variable symbols:	A lowercase letter (except those denoting constants)
Predicate symbols:	Uppercase letter, followed by zero or more letters
Function symbols:	Lowercase letter, followed by zero or more letters (except those denoting constants or variables)
Quantifier symbols:	$\forall$, $\exists$
Logical connectives:	$\neg, \wedge, \vee, \leftarrow$
Brackets:	(,)

In addition, we will sometimes employ the device of using subscripts to denote unique symbols (for example, $x_1, x_2, \ldots$ for a string of variables).

With this vocabulary, a *term* is simply a constant, variable or a functional expression (that is, a function applied to a tuple of terms). The following are all examples of terms: x, $fred$, $father(fred)$, $father(father(fred))$, $father(x)$.

These, however, are not terms: $\mathit{Likes}(\mathit{fred}, \mathit{bananas})$, $\mathit{Likes}(\mathit{fred}, \mathit{father}(\mathit{fred}))$, $\mathit{father}(\mathit{Likes}(\mathit{fred}, \mathit{bananas}))$. An *atomic formula*, sometimes simply called an *atom* is a predicate symbol applied to a tuple of terms. Thus, $\mathit{Likes}(\mathit{fred}, \mathit{bananas})$ $\mathit{Likes}(\mathit{fred}, \mathit{father}(\mathit{fred}))$, $\mathit{Likes}(x, \mathit{father}(x))$ are all examples of atoms. Finally, a *ground atomic formula* or a *ground atom* is an atom without any variables. Well-formed formulæ (wffs) are then formed using the following rules:

1. Any ground atomic formula is a wff;

2. If α is a wff then $\neg\alpha$ is a wff;

3. If α and β are wffs then $(\alpha \wedge \beta), (\alpha \vee \beta)$, and $(\alpha \leftarrow \beta)$ are wffs; and

4. If α is wff containing a constant c and $\alpha^{c/x}$ be the result of replacing one or more occurences of c with a variable x that does not appear in α. Then $\forall x \alpha^{c/x}$ and $\exists x \alpha^{c/x}$ are wffs.

Rules 1–3 are like their propositional counterparts (page 15). Rule 4 is new, and requires further explanation. It is the only way variables are introduced into a formula. As an example, take the following statement: $(\mathit{Human}(\mathit{fred}) \wedge \mathit{Likes}(\mathit{fred}, \mathit{bananas}))$. That this a wff follows from an application of Rules 1 and 3. The following formulæ are all wffs, following a single application of Rule 4:

$$\forall x(\mathit{Human}(x) \wedge \mathit{Likes}(\mathit{fred}, \mathit{bananas}))$$
$$\exists x(\mathit{Human}(x) \wedge \mathit{Likes}(\mathit{fred}, \mathit{bananas}))$$
$$\forall x(\mathit{Human}(\mathit{fred}) \wedge \mathit{Likes}(x, \mathit{bananas}))$$
$$\exists x(\mathit{Human}(\mathit{fred}) \wedge \mathit{Likes}(x, \mathit{bananas}))$$
$$\forall x(\mathit{Human}(x) \wedge \mathit{Likes}(x, \mathit{bananas}))$$
$$\exists x(\mathit{Human}(x) \wedge \mathit{Likes}(x, \mathit{bananas}))$$
$$\forall x(\mathit{Human}(\mathit{fred}) \wedge \mathit{Likes}(\mathit{fred}, x))$$
$$\exists x(\mathit{Human}(\mathit{fred}) \wedge \mathit{Likes}(\mathit{fred}, x))$$

A single application of Rule 4 therefore only introduces a single new variable. Subsequent applications will introduce more. For example, take the first formula above: $\forall x(\mathit{Human}(x) \wedge \mathit{Likes}(\mathit{fred}, \mathit{bananas}))$. The following wffs all result from applying Rule 4 to this statement:

$$\forall y \forall x(\mathit{Human}(x) \wedge \mathit{Likes}(y, \mathit{bananas}))$$
$$\exists y \forall x(\mathit{Human}(x) \wedge \mathit{Likes}(y, \mathit{bananas}))$$
$$\forall y \forall x(\mathit{Human}(x) \wedge \mathit{Likes}(\mathit{fred}, y))$$
$$\exists y \forall x(\mathit{Human}(x) \wedge \mathit{Likes}(\mathit{fred}, y))$$

As with propositional logic, it is acceptable to drop outermost brackets:

$$(\forall x(Ape(x) \leftarrow Human(x)) \wedge \exists x(Ape(x) \wedge \neg Human(x)))$$

can be written as:

$$\forall x(Ape(x) \leftarrow Human(x)) \wedge \exists x(Ape(x) \wedge \neg Human(x))$$

Clausal Form

We are now in a position to expand on the notion of clauses and literals, first introduced on page 25. Consider the conditional statement:

$$\forall x(Ape(x) \leftarrow Human(x))$$

Recall the following from page 24:

$$(\alpha \leftarrow \beta) \equiv (\alpha \vee \neg\beta)$$

This means:

$$\forall x(Ape(x) \leftarrow (Human(x))) \equiv \forall x(Ape(x) \vee \neg Human(x))$$

The term in brackets on right-hand side is an example of a *clause* in first-order logic. In general, formulæ consisting of universally-quantified clauses all look alike:[23]

$$\forall x_1 \forall x_2 \ldots (\alpha_1 \wedge \alpha_2 \ldots)$$

That is, they consist of a prefix that consists only of universally quantifiers, and each α_i, or clause, is a quantifier-free formula that looks like:

$$\alpha_i = (\beta_1 \vee \beta_2 \vee \ldots \beta_n)$$

where each β_j, or *literal*. Thus, as in propositional logic, a clause is a disjunction of literals. Each literal, however, is not a proposition, but is either an atomic formula (like $Ape(x)$, sometimes called a *positive* literal) or a negated atomic formula (like $\neg Human(x)$, sometimes called a *negative* literal). It is sometimes convenient to use $\forall\mathbf{x}$ to denote $\forall x_1 \forall x_2 \cdots$ and to adopt a set-based notation to represent a clausal formula: $\{\forall\mathbf{x}\alpha_1, \forall\mathbf{x}\alpha_2, \ldots\}$. Here it is understood that the formula stands for a conjunction of clauses. Often, the quantification is taken to be understood and left out. Further, individual clauses are themselves sometimes written as sets of literals:

$$\alpha_i = \{\beta_1, \beta_2, \ldots, \beta_n\}$$

Clausal forms are of particular interest, computationally speaking. The language of logic programs (usually written in the Prolog language). is, at least in its 'pure' form, equivalent to clausal-form logic. Here is an example:

[23] We will take a few liberties here by not including some brackets.

Logic Program in Prolog	Clausal Form
grandfather(X,Y):-	$\{\forall x \forall y \forall z (Grandfather(x,y) \vee$
father(X,Z),	$\neg Father(x,z) \vee$
parent(Z,Y).	$\neg Parent(z,y)),$
father(henry,jane).	$Father(henry,jane),$
parent(jane,john).	$Parent(jane,john)\}$

There are three clauses in this example. The first clause has three literals and the remainder have one literal each. Further, each clause has exactly one positive literal: such clauses are called *definite* clauses. More generally, clauses that contain *at most* one positive literal are called *Horn* clauses. From now on, we may sometimes write clauses in a lazy manner that is somewhere in between the syntax of Prolog and the true clausal form:

Logic Program in Prolog

```
grandfather(X,Y):- father(X,Z), parent(Z,Y).
```

Clausal Form

$$\{\forall x \forall y \forall z (Grandfather(x,y) \ \vee \ \neg Father(x,z) \ \vee \ \neg Parent(z,y))\}$$

Lazy Clausal Form

$$Grandfather(x,y) \leftarrow Father(x,z), Parent(z,y)$$

Skolem Functions and Constants

Skolemization refers to a process of replacing an existentially quantified variable in a formula by a new term; it is merely the process of providing a *name* for something that already exists. Whether the new term is a functional expression or a constant depends on where the existential quantifier appears in the formula. If it is preceded by one or more universal quantifiers, like:

$$\forall x_1 \ldots \forall x_n \exists y \ \alpha$$

then a single skolemization step replaces all occurences of y in α by the functional expression $\mathbf{f}(x_1, \ldots, x_n)$. Here $\mathbf{f}(\cdots)$ is a function symbol, called a *Skolem function*, that does not appear anywhere in the formula. Thus, skolemization of the formula $\forall x \exists y Likes(x,y)$ results in $\forall x Likes(x, \mathbf{f}(x))$. In general, if the existential quantifier appears in between some universal quantifiers:

$$\forall x_1 \ldots \forall x_{i-1} \exists y \forall x_{i+1} \cdots \forall x_n \ \alpha$$

then all occurences of y in α can be replaced by the functional expression $\mathbf{f}(x_1, \ldots, x_{i-1})$. A special case arises if the existential quantifier precedes zero or more universal quantifiers:

$$\exists y \forall x_1 \ldots \forall x_n \ \alpha$$

In this case, a single skolemization step replaces all occurences of y in α by a Skolem function of arity 0 (that is, a constant) $\mathbf{c}$. Here $\mathbf{c}$ is a constant symbol, called a *Skolem constant*, that does not appear anywhere in the formula. Thus, skolemization of the formula $\exists y \forall x Likes(x, y)$ results in $\forall x Likes(x, \mathbf{c})$. You can see that in both cases, a single step of Skolemization reduces the number of existential quantifiers in a formula ϕ by 1. Let us denote this single step by $s(\phi)$. It should be easy to see that repeatedly performing Skolemization steps (that is, $s(s(\cdots s(\phi))))$ will result in a formula with just universal quantifiers.

Normal Forms

Universally-quantified clausal forms are a special case of specific kind of normal form for first-order formula, which we are now able to present, having described the process of Skolemization. A formula is said to be in *prenex normal form* or PNF if all its quantifiers are in front. That is, the formula looks something like $Q_1 x_1 \ldots Q_n x_n \phi$, where the Q_i is either a $\forall$ or $\exists$. Further, if ϕ is a conjunction of disjunction of literals, then the formula is said to be in *conjunctive prenex normal form*. It can be shown that every first-order formula ϕ can be expressed by an equivalent one ϕ' in conjunctive PNF. Here is an example: suppose we want to find the conjunctive PNF for $\phi : (\exists x \forall y DisLikes(x, y) \wedge \forall x \exists y Likes(x, y))$. We first rename variables to give $\phi' : (\exists x \forall y DisLikes(x, y) \wedge \forall u \exists v Likes(u, v))$. We can then move the quantifiers to the left giving: $\phi'' : \exists x \forall y \forall u \exists v (DisLikes(x, y) \wedge Likes(u, v))$, which is in conjunctive PNF.

For reasons that will become apparent, we are further interested here only in formulae in conjunctive PNFs in which all the quantifiers are universal ones (that is, $\forall$). Such formulæare said to be in *Skolem normal form*, or SNF. A SNF can be obtained from the conjunctive PNF by applying the Skolemization process to eliminate $\exists$ quantifiers. For example, suppose we want to "Skolemize" the formula ϕ above. We first find the conjunctive PNF (ϕ'' above). Using the Skolemization procedure described earlier, we replace x by a Skolem constant and v by a Skolem function. The SNF of ϕ is $\phi^S : \forall y \forall u (Dislikes(\mathbf{c}, y) \wedge Likes(u, \mathbf{f}(u)))$.

We can now return to clausal forms. Here are the steps for converting a formula into a set of clauses in clausal form:

1. Rename variables to ensure there are no variables with the same names in different quantifiers;

2. Eliminate $\leftarrow$'s and iff's;

   - α_1 iff $\alpha_2 \Rightarrow (\alpha_1 \leftarrow \alpha_2) \wedge (\alpha_2 \leftarrow \alpha_1)$
   - $\alpha_1 \leftarrow \alpha_2 \Rightarrow \alpha_1 \vee \neg \alpha_2$

3. Move $\neg$'s inwards;

- $\neg(\exists \mathbf{X})\alpha \Rightarrow (\forall \mathbf{X})\neg\alpha$
- $\neg(\forall \mathbf{X})\alpha \Rightarrow (\exists \mathbf{X})\neg\alpha$
- $\neg(\alpha_1 \vee \alpha_2) \Rightarrow \neg\alpha_1 \wedge \neg\alpha_2$
- $\neg(\alpha_1 \wedge \alpha_2) \Rightarrow \neg\alpha_1 \vee \neg\alpha_2$
- $\neg\neg\alpha \Rightarrow \alpha$

4. Distribute $\vee$'s over $\wedge$'s;

   - $\alpha \vee (\alpha_1 \wedge \alpha_2) \Rightarrow (\alpha \vee \alpha_1) \wedge (\alpha \vee \alpha_2)$
   - Assuming **X** does not occur in α_1: $\alpha_1 \vee (\forall\alpha_2) \Rightarrow (\forall \mathbf{X})(\alpha_1 \vee \alpha_2)$
   - Assuming **X** does not occur in α_1: $\alpha_1 \vee (\exists\alpha_2) \Rightarrow (\exists \mathbf{X})(\alpha_1 \vee \alpha_2)$

5. Distribute $\forall$'s;

   - $(\forall \mathbf{X})(\alpha_1 \wedge \alpha_2) \Rightarrow (\forall \mathbf{X})\alpha_1 \wedge (\forall \mathbf{X})\alpha_2$

   At this stage, the PNF form is generated.

6. Skolemise existentially quantified variables;

   - $(\forall X_1)(\forall X_2)\ldots(\forall X_n)(\exists Y)\alpha(Y) \Rightarrow (\forall X_1)(\forall X_2)\ldots(\forall X_n)\alpha(f(X_1, X_2, \ldots X_n))$

   If further applications of step 6 are possible, then they should be carried out.

7. Rewrite as clauses by dropping universal quantifiers; and

8. Standardise variables apart.

Here are some statements from a book by Lewis Carroll, written in first-order logic:

$S_1 : \forall x(\mathit{Scented}(x) \leftarrow \mathit{Coloured}(x)) \wedge$

$S_2 : \forall x(\mathit{DisLike}(x) \leftarrow \neg\mathit{GrownOpen}(x)) \wedge$

$S_3 : \neg(\exists x(\mathit{GrownOpen}(x) \wedge \neg\mathit{Coloured}(x))) \wedge$

$S_4 : \neg\forall x((\mathit{DisLike}(x) \leftarrow \neg\mathit{Scented}(x)))$

You should find that these sentences, when converted by the steps above, give the following set of clauses:

$C_1 : \{\neg\mathit{Coloured}(x), \mathit{Scented}(x)\}$

$C_2 : \{\mathit{GrownOpen}(y), \mathit{DisLike}(y)\}$

$C_3 : \{\neg\mathit{GrownOpen}(z), \mathit{Coloured}(z)\}$

$C_4 : \{\neg\mathit{Scented}(\mathbf{c})\}$

$C_5 : \{\neg DisLike(\mathbf{c})\}$

Recall that representing a clause by a set $\{L_1, L_2, \ldots, L_k\}$, is just short-form for the disjunction $L_1 \vee L2 \vee \cdots L_k$; and the actual clausal form for the formula $F : S_1 \wedge S_2 \wedge S_3 \wedge S_4 \wedge S_5$ is $C : \forall x \forall y \forall z (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5)$. Is F equivalent to C? To answer this, we need to understand how meanings are assigned to first-order formulæ.[24]

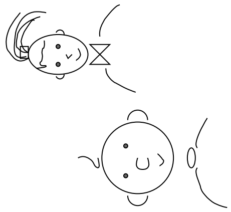
1.4.2 Semantics

As with propositional logic, the semantics of predicate logic is primarily concerned with interpretations, models, and logical consequence. Of these, it is only the notion of interpretation that requires a re-examination.

Interpretations

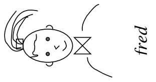
Recall that interpretations in propositional logic were simply assignments of *true* or *false* to propositional symbols. Matters are not as simple in predicate logic for two reasons. First, we have to deal with the additional complexity of expressions arising from a richer vocabulary. Thus, the truth-value (meaning) of $Likes(fred, bananas)$ will depend on the meanings of each of the symbols $Likes$, $fred$ and $bananas$. Second, we have to interpret sentences that contain quantifiers.

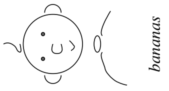
Informally, let us see what is required in constructing an interpretation that allows us to understand $Likes(fred, bananas)$. Before we begin, remove any pre-conceptions of what the words in the statement mean in English: for us they are simply a predicate symbol ($Likes$) and two constant symbols ($fred, bananas$) in some 'formal-world'. The first step then is to identify a domain of objects in the 'real-world'.



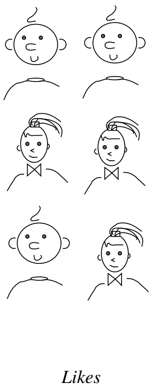
Next, we associate constant symbols in our formal-world to objects in the real-world (just to avoid any pre-conceptions, we have scrambled things a little bit):

[24] But the answer is "no": the problem, as you might have guessed, comes about because of the Skolemization step.





and associate the predicate symbols to a relation in the real-world:



We can now see that $Likes(fred, bananas)$ is *false* as the objects corresponding to the ordered pair $< fred, bananas >$ are not in the real-world relation represented by $Likes$.

Formally, an interpretation in predicate logic is a specification of:

1. A domain D;

2. A mapping of constants to elements in D;

3. A mapping of each n-argument predicate symbol to a relation on D^n, where $D^n = \{< d_1, \ldots, d_n > \mid d_i \in D\}$ is the n-fold Cartesian product; and

4. A mapping of each n-argument function symbol to a function from $D^n \rightarrow D$

You will also see sometimes that the term "structure" is used to describe what we have called an interpretation. In such cases, a distinction is made between the *vocabulary*, which consists of the constants, functions and predicate symbols; the and the *structure*, which consists of the domain D and the three mappings. We will continue to use "interpretation" to retain a similarity to propositional logic.

Given an interpretation, every atom—a predicate symbol with a tuple of terms as arguments—is assigned a truth-value according to whether the objects designated by the arguments are in the relation designated by the predicate symbol.

Well-formed formulæ in predicate logic consist of more than atoms. We also need rules for assigning truth-values to formulæ that contain logical connectives ($\neg, \wedge, \vee, \leftarrow$) and quantifiers ($\forall, \exists$). The semantics of the logical connectives in predicate logic are the same as those in propositional logic. Thus, as before, assigning meanings to formulæ with these connectives requires the use of the truth tables on page 17. For example, the formula $Human(fred) \wedge Likes(fred, bananas)$ is *true* only if the interpretation results in both the atoms $Human(fred)$ and $Likes(fred, bananas)$ being *true*. What though, of formulæ that contain quantified variables? The rules for these are:

1. Any wff $\forall x \alpha$ is *true* if and only if for every domain element that we can associate with x, α is *true*;

2. Any wff $\exists x \alpha$ is *true* if and only if for some domain element that we can associate with x, α is *true*.

Models and Logical Consequence

The meanings of these, and related concepts, are unchanged from propositional logic. Thus:

Models. Any interpretation that makes a wff *true* is called a model for that formula;

Validity and Unsatisfiability. A formula for which all interpretations are models is said to be valid. A formula for which none of the interpretations are a model is said to be unsatisfiable. A formula that has at least one model is said to be satisfiable;

Consequence. Given a conjunction of wffs Σ represented as the set $\{\beta_1, \ldots, \beta_n\}$ and a wff α if $\Sigma \models \alpha$ then every model for Σ is a model for α;

Deduction Theorem. Given a conjunction of wffs $\Sigma = \{\beta_1, \ldots, \beta_n\}$ and a wff α, $\Sigma \models \alpha$ if and only if $\Sigma - \{\beta_i\} \models (\alpha \leftarrow \beta_i)$. The proof is the same as was for that in the propositional case (*c.f.* theorem 6);

Equivalence. Given a pair of wffs α and β, if $\alpha \models \beta$ and $\beta \models \alpha$ then α and β are equivalent ($\alpha \equiv \beta$).

The following relations hold in the predicate logic (as usual, α, β are wffs):

$\neg(\alpha \vee \beta)$	$\equiv$	$\neg\alpha \wedge \neg\beta$	De Morgan's law
$\neg(\alpha \wedge \beta)$	$\equiv$	$\neg\alpha \vee \neg\beta$	De Morgan's law
$(\alpha \leftarrow \beta)$	$\equiv$	$(\alpha \vee \neg\beta)$	
$(\alpha \leftarrow \beta)$	$\equiv$	$(\neg\beta \leftarrow \neg\alpha)$	Conditional $\equiv$ Contrapositive
$\neg\forall x\alpha$	$\equiv$	$\exists x\neg\alpha$	
$\neg\exists x\alpha$	$\equiv$	$\forall x\neg\alpha$	
$\forall x\alpha$	$\equiv$	$\forall y\alpha^{x/y}$	Renaming of x by y
$\exists x\alpha$	$\equiv$	$\exists y\alpha^{x/y}$	Renaming of x by y
$\forall x\forall y\alpha$	$\equiv$	$\forall y\forall x\alpha$	
$\exists x\exists y\alpha$	$\equiv$	$\exists y\exists x\alpha$	
$\forall x(\alpha \wedge \beta)$	$\equiv$	$(\forall x\alpha \wedge \forall x\beta)$	Distributivity of $\forall$
$\exists x(\alpha \vee \beta)$	$\equiv$	$(\exists x\alpha \vee \exists x\beta)$	Distributivity of $\exists$
$\forall x\alpha$	$\models$	$\exists x\alpha$	
$(\forall x\alpha \vee \forall x\beta)$	$\models$	$\forall x(\alpha \vee \beta)$	
$\exists x(\alpha \wedge \beta)$	$\models$	$(\exists x \wedge \exists x\beta)$	
$\forall x\alpha^{y/\mathbf{f}(x)}$	$\models$	$\forall x\exists y\alpha$	Non-equivalence of Skolemized form
$\forall x\alpha^{y/\mathbf{c}}$	$\models$	$\exists y\forall x\alpha$	Non-equivalence of of Skolemized form

More on Normal Forms

We stated earlier that any first-order formula could be converted to a conjunctive prenex normal form, or conjunctive PNF. We further saw how a formula $\phi = Q_1x_1 \ldots Q_nx_n\phi_0(x_1, \ldots, x_n)$ could be "Skolemized" to give a formula ϕ^S in Skolem Normal Form, or SNF. The interest in SNFs lies in the following fact:

Theorem 15 *ϕ is satisfiable if and only if ϕ^S is satisfiable (you should be able to convince yourself that checking for satisfiability is equivalent to checking for logical consequence).*

Proof sketch: Recall that $\phi^S = s(s(\cdots s(\phi)))$ where $s(\cdot)$ denotes a single step of Skolemization. Now, it is sufficient to show that ϕ is satisfiable if and only if $s(\phi)$ is satisfiable (the full proof will follow by induction). We can also assume that ϕ is in conjunctive PNF. Now $s(\phi)$ results in either replacing a variable by a Skolem constant or by a Skolem function. Since the former is just a special case of the latter, we will just consider the case when $s(\phi)$ results in replacing a variable by a Skolem function. Let Q_i be the existential quantifier removed by the Skolemization step, and let $\psi(x_1, \ldots, x_i) = Q_{i+1} \ldots Q_n\phi_0(x_1, \ldots, x_n)$. Suppose $\forall x_1 \ldots \forall x_{i-1}\exists x_i\psi(x_1, \ldots, x_i)$ has some model M. Let $M_{\mathbf{f}}$ extend M by interpreting $\mathbf{f}$ in such a way that $M_{\mathbf{f}}$ is a model for $\psi(c_1, \ldots, c_{i-1}, \mathbf{f}(c_1, \ldots, c_{i-1}))$ for all possible values $c_1, \ldots, c_{i-1} \in M$ for the variables $x_1, \ldots, x_{i-1}$. Then, clearly, $M_{\mathbf{f}}$ is a model for $\forall x_1 \ldots \forall x_{i-1}\psi(x_1, \ldots, x_{i-1}, \mathbf{f}(x_1, \ldots, x_{i-1}))$. Now consider the converse. Suppose $\forall x_1 \ldots \forall x_{i-1}\psi(x_1, \ldots, x_{i-1}, \mathbf{f}(x_1, \ldots, x_{i-1}))$ has some model M then, it follows from the meaning of $\exists$ that M is also a model for $\forall x_1 \ldots \forall x_{i-1} \exists x_i \psi(x_1, \ldots, x_i)$. It follows therefore that ϕ is satisfiable if and only if $s(\phi)$ is satisfiable. □

We end this section on a note of caution: ϕ and $s(\phi)$ are not equivalent. That is, Skolemization does not preserve logical equivalence. A simple example should convince you of this. Let $\phi = \exists x First(x)$ and $s(\phi) = First(\mathbf{c})$. Clearly, we can find models for $\exists First(x)$ that are not models of $First(\mathbf{c})$.

Herbrand Interpretations

The 4-step specification above makes an interpretation in predicate logic much more elaborate than its counterpart in propositional logic (which was simply an assignment of *true* or *false* to propositions). The reference to 'real-world' objects, relations, and functions adds a further degree of complexity: how is all this to be conveyed to an automated procedure? In fact, many of these problems can be side-stepped by confining attention only to a domain that consists solely of formal symbols. Called the *Herbrand universe* (U_L), this is simply all the ground (or variable-free) terms that can be constructed using the constants and function symbols available in a first order language L. Consider as example a language that consists of:

Constant symbol:	$zero$
Predicate symbol:	$Nat/1$
Function symbols:	$pred, succ$

The Herbrand universe U_L in this instance consists of terms like $zero$, $pred(zero)$, $succ(zero)$, $pred(succ(zero))$, $succ(pred(zero))$ and so on. The *Herbrand base* B_L is the set of all ground atoms that can be constructed using the predicate symbols and terms from the Herbrand universe U_L. Here, the Herbrand base B_L consists of atoms like $Nat(zero), Nat(pred(zero)), Nat(succ(zero)), Nat(pred(succ(zero)))$ and so on. A Herbrand interpretation I_L is—quite like the propositional case—simply an assignment of *true* to some subset of B_L and *false* to the rest. In fact, it is common practice to associate 'Herbrand interpretation' only with the subset assigned *true*: it being understood that all other atoms in the Herbrand base are assigned *false*. Thus, $\{Nat(zero)\}$ is an I_L that assigns *true* to $Nat(zero)$ and *false* to all other atoms in B_L.

Herbrand Models

Since a model is an interpretation that makes a well-formed formula *true*, a *Herbrand model* M_L is simply a Herbrand interpretation I_L that makes a well-formed formula *true*. Let us return to the example presented earlier:

Constant symbol:	$zero$
Predicate symbol:	$Nat/1$
Function symbols:	$pred, succ$

Recall from page 68, that:

Herbrand universe (U_L):	$\{zero, pred(zero), succ(zero), pred(succ(zero)), \ldots\}$
Herbrand base (B_L):	$\{Nat(zero), Nat(pred(zero)), Nat(succ(zero)), \ldots\}$

Further, a Herbrand interpretation is simply a subset of the Herbrand base containing all atoms that are *true*. Thus, $I_1 = \{Nat(zero)\}$ is a Herbrand interpretation in which $Nat(zero)$ is *true* and all other atoms in the Herbrand base are are *false*. We can now examine whether I_1 is a Herbrand model for the formula:

$$\Sigma_1 : Nat(zero) \ \wedge \ \forall x(Nat(succ(x)) \leftarrow Nat(x))$$

Being a conjunctive expression, we require I_1 to be a Herbrand model for both $Nat(zero)$ and $\forall x(Nat(succ(x)) \leftarrow Nat(x))$. I_1 is clearly a model for $Nat(zero)$ as this atom is assigned *true* in the interpretation. But what about the conditional? The rule for the universal quantifier (page 66) dictates that the conditional statement is *true* if it is *true* for every element that the variable x can be associated with. In other words, the interpretation is a model for every element that x can be associated with. In the Herbrand world, x can be associated with any element of the Herbrand universe ($zero, succ(zero), pred(zero)$ and so on). Suppose x was associated with $zero$. Then we would require I_1 to be a model for $Nat(succ(zero)) \leftarrow Nat(zero)$. Since I_1 assigns $Nat(succ(zero))$ to *false* and $Nat(zero)$ to *true*, I_1 is not a model for $Nat(succ(zero)) \leftarrow Nat(zero)$ (line 2 in the truth-table for the conditional on page 17). Thus, I_1 is not a Herbrand model for $\forall x(Nat(succ(x)) \leftarrow Nat(x))$ and in turn for Σ_1. Consider, on the other hand, the formula:

$$\Sigma_2 : Nat(zero) \ \wedge \ \forall x(Nat(x) \leftarrow Nat(pred(x)))$$

As before, suppose x was associated with $zero$. The conditional then becomes $Nat(zero) \leftarrow Nat(pred(zero))$. With interpretation I_1, $Nat(zero)$ is *true* and $Nat(pred(zero))$ is *false*. I_1 is therefore a model for this formula (line 3 in the truth table for the conditional). All other associations for x result in both sides of the conditional being *false* and I_1 being a model for each such formula (line 1 in the truth table). Thus, I_1 makes $\forall x(Nat(x) \leftarrow Nat(pred(x)))$ *true* for every element that x can be associated with, and is a model for it and in turn for Σ_2.

Herbrand models are particularly relevant to the study of clausal forms (page 60). Recall that these are conjunctions of clauses, each of which contains only universally quantified variables and consists of a disjunction of literals. Both Σ_1 and Σ_2 above can be written in clausal form:

$${\Sigma_1}' : Nat(zero) \ \wedge \ \forall x(Nat(succ(x)) \vee \neg Nat(x))$$

$${\Sigma_2}' : Nat(zero) \ \wedge \ \forall x(Nat(x) \vee \neg Nat(pred(x)))$$

The *ground instantiation* of a clausal formula is the conjunction of ground (variable-free) clauses that result by replacing variables with terms from the Herbrand universe. For example, the ground instantiation of ${\Sigma_2}'$ is:

$$\begin{array}{lll} \mathcal{G}({\Sigma_2}'): & Nat(zero) & \wedge \\ & (Nat(zero) \vee \neg Nat(pred(zero))) & \wedge \\ & (Nat(pred(zero)) \vee \neg Nat(pred(pred(zero)))) & \wedge \\ & (Nat(succ(zero)) \vee \neg Nat(pred(succ(zero)))) & \wedge \\ & \ldots & \end{array}$$

You can therefore think of the ground instantantiation as making explicit the meaning of the universal quantifier $\forall$. Now, it should be clear that a Herbrand interpretation will determine the truth-value for all clauses in the ground instantiation of a clausal formula.

We present another example illustrating Herbrand models. Consider the following program P:

$$likes(john, X) \ \leftarrow \ likes(X, apples)$$

$$likes(mary, apples) \ \leftarrow$$

Suppose the language $\mathcal{L}$ contained no symbols other than those in P. Then, $\mathcal{B}(P)$ is the set $\{likes(john, john), likes(john, apples), likes(apples, john), likes(john, mary),$ $likes(mary, john), likes(mary, apples), likes(apples, mary), likes(mary, mary),$ $likes(apples, apples)\}$. Now, $\{likes(mary, apples), likes(john, mary)\}$ is a subset of $\mathcal{B}(P)$, and is a Herbrand interpretation. Moreover, it is also a Herbrand model for P. Similarly, $\{likes(mary, apples), likes(john, mary), likes(mary, john)\}$ is also a model for P. The ground instantiation $\mathcal{G}(P)$ for this program is:

$$likes(john, john) \ \leftarrow likes(john, apples)$$

$$likes(john, mary) \ \leftarrow likes(mary, apples)$$

$$likes(john, apples) \ \leftarrow likes(apples, apples)$$

$$likes(mary, apples) \ \leftarrow$$

It can be verified[25] that the interpretation $\{likes(mary, apples), likes(john, mary)\}$ is a model for the $\mathcal{G}(P)$ above.

The importance of Herbrand models for clausal formulæ stems from the following property:

Theorem 16 *A clausal formula Σ has a model if and only if its ground instantiation $\mathcal{G}(\Sigma)$ has a Herbrand model.*

Proof: $\Rightarrow$: Suppose Σ has a model M. Then we define the following Herbrand interpretation I as follows. Let P be an n-ary predicate symbol occurring in Σ. Then we define the function I_P from U_L^n to $\{T, F\}$ as follows: $I_P(t_l, \ldots, t_n) = T$ if $P(t_1, ..., t_n)$ is true under M, and $I_P(t_1, ..., t_n) = F$ otherwise. It can easily be shown that $I = \cup_{P \in \Sigma} I_P$ is a Herbrand model of Σ.

$\Leftarrow$: This is obvious (a Herbrand model is a model). $\Box$

[25] EXERCISE.

In other words, there must be *some* assignment of truth-values to atoms in the Herbrand base that makes all clauses in Σ *true*. In the example above, the Herbrand interpretation $I_1 = \{Nat(zero)\}$ that assigns $Nat(zero)$ to *true* and everything else to *false*, is clearly a model for $\mathcal{G}({\Sigma_2}')$. Therefore, from the property stated here, we can say that ${\Sigma_2}'$ has a model.

If we are dealing only with a *definite clausal formula*— a clausal formula in which all clauses have exactly one positive literal (${\Sigma_1}'$ and ${\Sigma_2}'$ are both of this type)—then more is known about the Herbrand models of the formula. Recall that a Herbrand model is nothing more than a set of ground atoms, which when assigned *true*, make the formula *true*.

1.4.3 From Datalog to Prolog

The statement *"Any animal that has hair is a mammal"* can be written as a clause using monadic predicates (*i.e.* predicates with arity 1):

$$\forall X\ \mathit{is\_mammal(X)} \leftarrow \mathit{has\_hair(X)}$$

Usually clauses are written without explicit mention of the quantifiers:

$$\mathit{is\_mammal(X)} \leftarrow \mathit{has\_hair(X)}$$

$$\mathit{is\_mammal(X)} \leftarrow \mathit{has\_milk(X)}$$

$$\mathit{is\_bird(X)} \leftarrow \mathit{has\_feathers(X)}$$

...

Datalog

Datalog is a subset of the language of first order language; it has all the components of first order logic (variables, constants and recursion), except functions. A Datalog "expert" system will encode these rules using monadic predicates as:

```
is_mammal(X) :- has_hair(X).
is_mammal(X) :- has_milk(X).
is_bird(X) :- has_feathers(X).
is_bird(X) :- can_fly(X), has_eggs(X).
is_carnivore(X) :- is_mammal(X), eats_meat(X).
is_carnivore(X) :- has_pointed_teeth(X), has_claws(X), has_pointy_eyes(X).
cheetah(X) :- is_carnivore(X), has_tawny_colour(X), has_dark_spots(X).
tiger(X) :- is_carnivore, has_tawny_colour(X), has_black_stripes(X).
penguin(X) :- is_bird(X), cannot_fly(X), can_swim(X).
```

Now here are some statements[26] particular to animals:

```
has_hair(peter).              fat(peter).
has_green_eyes(peter).        has_tawny_colour(peter).
eats_meat(peter).             has_black_stripes(peter).
has_milk(bob).                eats_meat(bob)
has_tawny_colour(bob).        has_dark_spots(bob).
can_fly(bob).
```

[26] EXERCISE: What are the logical consequences of all the clauses?

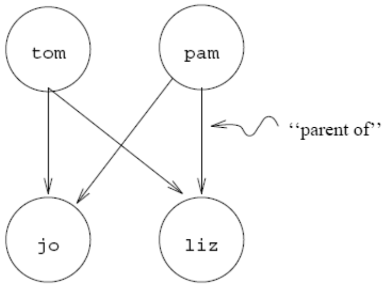


Figure 1.11: Graph representing 'parent of' relation.

However, monadic predicates: not expressive enough. While monadic predicates lets us make statements like *"Every son has a parent"*:

$$\forall X \exists Y \; parent(Y) \leftarrow son(X)$$

for more complex relationships, we will need predicates of arity > 1. Usually, relationships can be described pictorially by a directed acyclic graph (DAG) as in Figure 1.11 The parent-child relation could also be specified as a set of ordered pairs $< X, Y >$, or, as a set of definite clauses

$$\begin{array}{l} parent(tom, jo) \leftarrow \\ parent(pam, jo) \leftarrow \\ parent(tom, liz) \leftarrow \\ parent(pam, liz) \leftarrow \end{array}$$

Consider the *predecessor* relation, namely, all ordered tuples $< X, Y >$ *s.t.* X is an ancestor of Y. This set will include Y's parents, Y's grandparents, Y's grandparents' parents, etc.

$$\begin{array}{lcl} pred(X, Y) & \leftarrow & parent(X, Y) \\ pred(X, Z) & \leftarrow & parent(X, Y), parent(Y, Z) \\ pred(X, Z) & \leftarrow & parent(X, Y1), parent(Y1, Y2), parent(Y2, Z) \\ \ldots & & \end{array}$$

As can be seen through this example, variables and constants are not enough: we need *recursion*:

$\forall X, Z$ X is a predecessor of Z if
1. X is a parent of Z; or
2. X is a parent of some Y, and Y is a predecessor of Z

The predecessor relation is thus

$$\begin{array}{lcl} pred(X, Y) & \leftarrow & parent(X, Y) \\ pred(X, Z) & \leftarrow & parent(X, Y), pred(Y, Z) \end{array}$$

and can be pictorially depicted as in 1.12

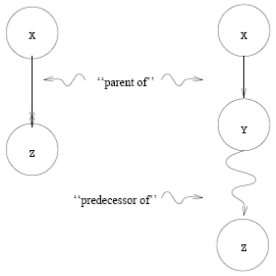


Figure 1.12: The predecessor relation.

Prolog = Predicates + Variables + Constants + Functions

Datalog (first order logic without functions) is however not expressive enough. To express arithmetic operations, lists of objects, etc. it is not enough to simply allow variables and constants as terms. We will also need *function* symbols as supported in Prolog.

Consider Peano's postulates for the set of natural numbers $\mathcal{N}$.

1. The constant 0 is in $\mathcal{N}$
2. if X is in $\mathcal{N}$ then $s(X)$ is in $\mathcal{N}$
3. There are no other elements in $\mathcal{N}$
4. There is no X in $\mathcal{N}$ $s.t.$ $s(X) = 0$
5. There are no X, Y in $\mathcal{N}$ $s.t.$ $s(X) = s(Y)$ and $X \neq Y$

We can write a definite clause definition using 1 constant symbol and 1 unary function symbol for enumerating the elements of $\mathcal{N}$:

$$natural(0) \leftarrow$$
$$natural(s(X)) \leftarrow natural(X)$$

The elements of $\mathcal{N}$ can be now generated by asking:

natural(N)?

Prolog also supports lists. Lists are simply collections of objects. For e.g. $1, 2, 3 \ldots$ or $1, a, dog, \ldots$. Lists are defined as follows:

1. The constant *nil* is a list
2. If X is a term, and Y is a list then $.(X, Y)$ is a list

So the list $1, 2, 3$ is represented as:

$$.(1,.(2,.(3,nil)))$$

Usually logic programming systems use a "[" "]" notation, in which the constant *nil* is represented as [] and the list 1, 2, 3 is [1, 2, 3]. In this notation, the symbol | is used to separate a list into a "head" (the elements to the left of the |) and a "tail" (the list to the right of the |). Thus:

List	Represented as	Values of variables
$[1,2,3]$	$[X\|Y]$	$X=1,Y=[2,3]$
$[[1,2],3]$	$[X\|Y]$	$X=[1,2],Y=[3]$
$[1]$	$[X\|Y]$	$X=1,Y=[]$
$[1\|2]$	$[X\|Y]$	$X=1,Y=2$
$[1]$	$[X,Y]$	
$[1,2,3]$	$[X,Y\|Z]$	$X=1,Y=2,Z=[3]$

1.4.4 Lattice of Herbrand Models

The discussion in this section is more or less similar to the discussion in Section 1.3.8 and the reader is referred to the proofs in that Section for proofs of most statements that will be made in this section. The only difference is that while Section 1.3.8, in this section, we will talk about Herbrand models.

For a definite clausal formula, it can be shown that the set H of Herbrand models is a complete lattice ordered by set-inclusion (that is, for Herbrand models $M1, M2 \in H$, $M1 \preceq M2$ if and only if $M1 \subseteq M2$), and with the binary operations of $\cap$ and $\cup$ as the glb and lub respectively. Recall that a complete lattice has a unique least upper bound and a unique greatest lower bound. Since we are really talking about sets that are ordered by set-inclusion, this means that there is a a unique *smallest* one (the size being measured by the number of elements). This is called the *minimal model* of the formula, and it can be shown that this must be the intersection of all Herbrand models for the formula.

In the example above, ${\Sigma_2}'$ has several Herband models ($\{Nat(zero)\}$ and $\{Nat(zero), Nat(pred(zero))\}$ are two examples). Of these $\{Nat(zero)\}$ is the smallest, and is the minimal model. There is an important result relating a definite clausal formula Σ, its minimal model $\mathcal{MM}(\Sigma)$ and the ground atoms that are logical consequences of Σ:

Theorem 17 *If α is a ground atom then $\Sigma \models \alpha$ if and only if $\alpha \in MM(\Sigma)$.*

Here $MM(\cdot)$ denotes the minimal model. Thus, the minimal model of a definite clausal formula is identical to the set of all ground atoms logically implied by that formula. Thus, the minimal model provides, in effect, denotes the meaning (or semantics) of the formula. The proof of this theorem follows nearly from theorem 12 that was proved earlier.

We can envisage a procedure for enumerating the Herbrand models of a formula. Consider the powerset of the Herbrand base of the formula. Now, we know that this powerset ordered by $\subseteq$ necessarily forms a complete lattice, with binary operations $\cap$ and $\cup$. Some subset of this powerset is the set of all Herbrand models, which we know is also a lattice ordered by $\subseteq$ with the same binary operations. So, the model lattice is a sublattice of the lattice obtained from the powerset of the Herbrand base. Suppose now we start at some point s in this sublattice, and we move to a new point that consists only of those ground atoms of the formula made true by the model s. Let us call these atoms s_1. Then, a little thought should convince you that s_1 is also a member of the sublattice of Herbrand models. Repeating the process with s_2 we can move to models s_2, s_3 and so on. Will this procedure converge eventually on the minimal model? Not necessarily, since we could end up moving back-and-forth between points of the sub-lattice. (When will this happen, and how can we ensure that we do converge on the minimal model?).

A slightly more general process can be formalised as the application of a function T_P that, for a clausal formula P, generates an interpretation (not necessarily a model) from another. That is:

$$I_{k+1} = T_P(I_k)$$

where

$$T_P(I) = \{a : a \leftarrow \mathit{body} \in \mathcal{G}(P) \text{ and } \mathit{body} \in I\}$$

where $\mathcal{G}(P)$ is the ground instantiation of P as before. It can be shown that T_P is both monotonic and continuous on the complete lattice obtained by ordering the powerset of the Herbrand base by $\subseteq$. So, we know from the Knaster-Tarski Theorem mentioned on page 11, that there must be a least fixpoint for T_P in this lattice. We can prove that the procedure of obtaining I_{k+1} from application of T_P to I_k will yield that fixpoint, and further, that this fixpoint will be the minimal model. As an inference procedure though, it is not really very practical: especially if all we needed to do is check if a particular atom was a logical consequence. It gets worse if the minimal model is not finite, in which case the procedure may not terminate in a finite number of steps. For all these reasons, we will need to do better.

1.4.5 Inference

Consider the following set of clauses S:

$$\begin{array}{l}
likes(john, flowers) \leftarrow \\
likes(mary, food) \leftarrow \\
likes(mary, wine) \leftarrow \\
likes(john, wine) \leftarrow \\
likes(john, mary) \leftarrow \\
likes(paul, mary) \leftarrow
\end{array}$$

If you entered these clauses into a program capable of executing logic programs (some implementation of Prolog), and asked:

$$likes(john, X)?$$

you will get a number of answers:

$$X = flowers$$
$$X = wine$$
$$X = mary$$

On the other hand, if the query were

$$likes(john, X), likes(mary, X)?$$

the answer should be:

$$X = wine$$

How this works will be examined in shortly. For now, consider *likes(john,X)?*. An intuitive procedure will be:

1. Start search from 1^{st} clause
2. Search for any clause whose head has predicate $likes/2$, and 1^{st} argument is $john$
3. If no clause is found *return* otherwise *goto* 4
4. X is associated ("instantiated") with the 2^{nd} argument of the head literal, the clause position marked, and the value associated with X is output
5. Start search from clause marked, and *goto* 2

As in the propositional case, we will only be concerned here with the rule of resolution. In a broad sense, this remains similar to its propositional counterpart (page 29: it applies to clauses with a pair of complementary literals, and the result (or resolvent) is a clause with the complementary pair removed. However the intricacies of predicate logic require a bit more care. Take the following pair of conditionals (and their clausal forms):

Conditional	Clausal Form
$\forall x(Ape(x) \leftarrow Human(x))$	$\forall x(Ape(x) \vee \neg Human(x))$
$Human(fred) \leftarrow Human(father(fred))$	$Human(fred) \vee \neg Human(father(fred))$

For resolution to apply, we require the clausal forms to contain a pair of complementary literals. We nearly do have such a pair: $\neg Human(x)$ in the first clause and $Human(fred)$ in the second. It is apparent that if variable x in the first clause were to be restricted to the term $fred$, then we would indeed have a complementary pair, and the resolvent is:

Resolvent	Clausal Form
$Ape(fred) \leftarrow Human(father(fred))$	$Ape(fred) \vee \neg Human(father(fred))$

A single resolution step in predicate logic thus involves 'substituting' terms for variables so that a complementary pair of literals results. Here, such a pair would result if we could somehow 'match' the literals $Human(x)$ and $Human(fred)$. The resulting mapping of variables to terms is called the *unifier* of the two literals. Thus, mapping x to $fred$ is a unifier for the literals $Human(x)$ and $Human(fred)$.

Substitution

More generally, a *substitution* is a mapping from variables to terms that is usually denoted as $\theta = \{v_1/t_1, v_2/t_2, \ldots, v_n/t_n\}$. Applying a substition θ to a well-formed formula α results in a *substitution instance*, usually denoted by $\alpha\theta$. Thus, applying the substitution $\theta = \{x/fred\}$ to $\alpha : \forall x(Ape(x) \vee \neg Human(x))$ results in the substitution instance $\alpha\theta : (Ape(fred) \vee \neg Human(fred))$. We usually require substitutions to have the following properties:

1. They should be *functions*. That is, each variable to the left of the / should be distinct. Thus, $\{x/fred, x/bill\}$ is not a legal substitution; and

2. They should be *idempotent*. That is, each term to the right of the / should not contain a variable that appears to the left of the /. Thus, $\{x/father(x)\}$ is not a legal substitution. This test is sometimes called the "occurs-check". The occur-check disallows self-referential bindings such as $X/f(X)$. However, the temptation to omit the occur-check in unification algorithms is very strong, owing to the high processing cost of including it; it is the only test in the comparison cycle which has to scrutinize the inner contents of terms, whereas all other tests examine only the terms' principal (outermost) symbols.

A pair of substitutions can be *composed* ('joined together'). For example, composing $\{x/father(y)\}$ with $\{y/fred\}$ results in $\{x/father(fred)\}$. In general, the result of composing substitutions

$$\theta_1 = \{u_1/s_1, \ldots, u_m/s_m\}$$

$$\theta_2 = \{v_1/t_1, \ldots, v_n/t_n\}$$

is (this may not be a legal substituition):

$$\theta_1 \circ \theta_2 = \{u_1/s_1\theta_2, \ldots, u_m/s_m\theta_2\} \cup \{v_i/t_i | v_i \notin \{u_1, \ldots, u_m\}\}$$

Theorem 18 *If α is a universally quantified expression that is not a term (i.e., a literal or a conjunction or disjunction of literals), and θ is a substitution, then the following holds: $\alpha \models \alpha\theta$. For example, $P(x) \vee \neg Q(y) \models P(a) \vee \neg Q(y)$, where we have used the substitution $\{x/a\}$.*

Proof sketch: The proof for this example is easy: suppose I is a model, with domain D, of $P(x) \vee \neg Q(y)$. Then for all $d_1 \in D$, and for all $d_2 \in D$, $I_P(d_1) = T$ or $I_Q(d_2) = F$. Suppose a is mapped to domain element d by I, then for all $d \in D$, $I_P(d) = T$ or $I_Q(d) = F$. Hence I is a model of $P(a) \vee \neg Q(y)$. It is clear that for different α or θ, a similar proof can always be given. Hence always $\alpha \models \alpha\theta$. □

Unifiers

We are now in a position to state more formally the notion of unifiers. To say that a substitution θ is a unifier for formulæ α_1 and α_2 means $\alpha_1\theta = \alpha_2\theta$. However, there can be many unifiers. For example, the formulæ $\alpha_1 : \forall x \forall z Parent(father(x), z)$ and $\alpha_2 : \forall y Parent(y, fred)$ have as unifiers $\theta_1 = \{x/fred, y/father(fred), z/fred\}$ and $\theta_2 = \{y/father(x), z/fred\}$. In the first case $\alpha_1\theta_1 = \alpha_2\theta_1 = Parent(father(fred), fred)$; and in the second case $\alpha_1\theta_2 = \alpha_2\theta_2 = \forall x Parent(father(x), x)$. Notice that θ_2 is, in some sense, more 'general' than θ_1 as it imposes less severe constraints on the variables. There is, in fact, a *most general unifier* (or mgu) for a pair of formulæ. The substitution θ is a most general unifier for α_1 and α_2 if and only if:

1. $\alpha_1\theta = \alpha_2\theta$ (that is, θ is a unifier for α_1 and α_2); and
2. For any other unifier σ for α_1 and α_2, there is a substitution μ such that $\sigma = \theta \circ \mu$ (that is, $\alpha_1\sigma$ is a substitution instance of $\alpha_1\theta$).

In the example just shown, θ_2 is the most general unifier.

Returning now to resolution, we can state the main steps involved for a pair of clauses C_1 and C_2:

1. Rename all variables in clause C_2 so that they cannot be confused with those in C_1 (for the variables in C_2 are independent of those in C_1 and the renamed clause is equivalent to C_2). This is sometimes called "standardising the clauses apart";
2. Identify complementary literals and see if an mgu exists;
3. Apply mgu and form the resolvent C.

Here is an example:

Formula	Clausal Form
$C_1 : \forall x(Ape(x) \leftarrow Human(x))$	$\forall x(Ape(x) \vee \neg Human(x))$
$C_2 : \forall x(Human(x) \leftarrow Human(father(x)))$	$\forall x(Human(x) \vee \neg Human(father(x)))$

The 3 steps above are:

1. Standardise apart. The two clauses are now:

$$C_1 : \forall x(Ape(x) \vee \neg Human(x))$$
$$C_2 : \forall y(Human(y) \vee \neg Human(father(y)))$$

2. Identify complementary literals and mgu. It is evident that $\neg Human(x)$ in C_1 and $Human(y)$ in C_2 are complementary. Their mgu is $\theta = \{x/y\}$;

3. Apply mgu and form resolvent. The resolvent C is as shown below:

$$C : \forall x(Ape(x) \vee \neg Human(father(x)))$$

As with propositional logic, the set-based notation used for clauses (page 60) allows us to present resolution in a compact (algebraic) form:

$$R = (C_1 - \{L\})\theta \cup (C_2 - \{M\})\theta$$

The difference to propositional logic is, of course, the appearance of θ, the mgu of literals L and $\neg M$. In fact, there is another problem that we have avoided. Suppose our clauses C_1 and C_2 are $C_1 : \forall x \forall y(Human(x) \vee Human(y))$ and $C_2 : \forall u \forall v(\neg Human(u) \vee \neg Human(v))$. Now it is clear that $\{C_1, C_2\}$ is unsatisfiable. But, unfortunately, we will not be able to get to the empty clause $\Box$ using resolution as we have just described it. Here is one possible resolvent: $R : \forall y \forall v(Human(y) \vee \neg Human(v))$. In fact, every possible resolvent of the two clauses will contain two literals, as will resolvents using those resolvents, and so on. What we really want to do is to eliminate redundant literals in any clause. For example, C_1 should really just be $\forall x Human(x)$ and C_2 should really just be $\forall u Human(u)$. The procedure that removes redundant literals in this manner is called *factoring*.

Factoring

Formally, if C is a clause, $L_1, \ldots, L_n (n \geq 1)$ some unifiable literals from C, and θ an mgu for the set $\{L_1, \ldots, L_n\}$, then the clause obtained by deleting $L_2\theta, \ldots, L_n\theta$ from $C\theta$ is called a *factor* of C. For example, $Q(a) \vee P(f(a))$ is a factor of the clause $\neg Q(a) \vee P(f(a)) \vee P(y)$ using $\{y/f(a)\}$ as an mgu for $\{P(f(a)), P(y)\}$. Also, $Q(x) \vee P(x, a)$ is a factor of $Q(x) \vee Q(y) \vee Q(z) \vee P(z, a)$.

Operationally, it finds a substitution that unifies one or more literals in a clause, and retains only a single copy of the unified literals. Semantically speaking, a literal L is redundant in a clause C, if it is equivalent to a clause without that literal. That is $C - \{L\} \equiv C$. Note that every non-empty clause C is a factor of C itself, using the empty substitution $\emptyset$ as mgu for one literal in C. It can easily be shown if C' is a factor of C, then $C \models C'$. We leave this to the reader to prove[27] From now on, we will assume that this elimination procedure has been executed on clauses, and we are only dealing with their "factors".

[27] Exercise.

Resolution

The rule of resolution remains sound for clauses in the predicate logic. That is, if C_1 and C_2 are clauses and R is a resolvent, then $\{C_1, C_2\} \models R$. The presence of variables and substitutions makes the proof of this a little more involved.

Theorem 19 *Suppose R is the result of resolving on literal L in C_1 and M in C_2. Let θ be the most general unifier of L and $\neg M$ that is used to obtain R. Then, the soundness of a single step of resolution means $\{C_1, C_2\} \models (C_1 - \{L\})\theta \cup (C_2 - \{M\})\theta$.*

Proof: Let M be a model for C_1 and C_2. Now, we know that either (a) $L\theta$ is true and $M\theta$ is false in M; or (b) $L\theta$ is false and $M\theta$ is true in M. Suppose the former. Since M is a model for C_2, it is a model for $C_2\theta$ (based on theorem 18). Therefore, at least one other literal $(C_2 - \{M\})\theta$ must be true in M. In other words, M is a model for $(C_1 - \{L\})\theta \cup (C_2 - \{M\})\theta$. Case (b) similarly results in M being a model for $(C_1 - \{L\})\theta$ and hence for R. So, a single resolution step is sound - the soundness of a proof consisting of several resolutions steps can be shown quite easily using the technique of induction. □

Recall the second property of resolution from propositional logic, namely that of refutation-completeness. In other words, if a formula (or a set of formulæ) is inconsistent, then the empty clause □ is derivable by the use of resolution. This property continues to hold for resolution in first-order logic. But before we look at that, we revisit an important result.

1.4.6 Subsumption Revisited

Recall that in propositional logic, a clause C subsumed a clause D if $C \subseteq D$. In first-order logic, this generalises as follows. A clause C *subsumes* a clause D if there is some substitution θ such that $C\theta \subseteq D$. What does this mean? It means that after applying the substitution θ to C, every literal in C appears in D. Here are a pair of clauses C and D such that C subsumes D:

$$C : Primate(x) \leftarrow Ape(x)$$

$$D : Primate(Henry) \leftarrow Ape(Henry), Human(Henry)$$

Here, a substitution of $\theta = \{x/Henry\}$ applied to C makes $C\theta \subseteq D$. In general:

Theorem 20 *If C and D are clauses such that $C\theta \subseteq D$ for some substitution θ, then $C \models D$.*

Proof: Since C is a universally quantified formula, by theorem 18, we must have $C \models C\theta$. Also, since clauses are disjunctions of literals and $C\theta \subseteq D$, clearly, $C\theta \models D$ and the result follows. □

However, unlike propositional logic, the reverse does not hold. That is, $C \models D$ does not necessarily mean that C subsumes D. Here is an example of this:

$$C : Human(x) \leftarrow Human(father(x))$$

$$D : Human(y) \leftarrow Human(father(father(y)))$$

With a little thought (let us not get too entangled in the species problem here), you should be able to convince yourself that $C \models D$. But you will find it impossible to find a substitution θ that will make $C\theta \subseteq D$. What makes the difference to the propositional case? The difference between implication and subsumption in first-order logic arises because of self-recursive clauses of the kind shown: a short, but influential paper by Georg Gottlob shows that it is indeed only the self-recursive case that results in the difference.

1.4.7 Subsumption Lattice over Atoms

The subsumption relation is an example of a quasi-order. Let us take the simple case of definite clauses with a single literal (that is, atoms). Consider the set $\mathcal{A}$ of all atoms in some language, and $\mathcal{A}^+ = \mathcal{A} \cup \{\top, \bot\}$. Let the binary relation $\succeq$ be such that:

- $\top \succeq \mathbf{l}$ for all $\mathbf{l} \in \mathcal{A}^+$
- $\mathbf{l} \succeq \bot$ for all $\mathbf{l} \in \mathcal{A}^+$
- $\mathbf{l} \succeq m$ iff there is a substitution θ such that $\mathbf{l}\theta = m$, for $\mathbf{l}, \mathbf{m} \in \mathcal{A}$

We will represent a list of elements $e_1, \ldots, e_n$ as the(as the language Prolog does) by $[e_1, \ldots, e_n]$, and let $\mathbf{l} = Mem(x, [x, y])$ and $\mathbf{m} = Mem(1, [1, 2])$ then $\mathbf{l} \succeq \mathbf{m}$ with $\theta = \{x/1, y/2\}$. It is easy to see that $\succeq$ is a quasi-order over $\mathcal{A}^+$: clearly $\mathbf{l} \succeq \mathbf{l}$, with the empty substitution $\theta = \emptyset$ (that is, $\succeq$ is reflexive). Now, let $\mathbf{l} \succeq \mathbf{m}$ and $\mathbf{m} \succeq \mathbf{l}$. That is, there are some substitutions θ_1 and θ_2 such that $\mathbf{l}\theta_1 = \mathbf{m}$ and $\mathbf{m}\theta_2 = \mathbf{l}$. That is, $(\mathbf{l}\theta_1) \circ \theta_2 = n$. With $\theta = \theta_1 \circ \theta_2$ it follows that $\mathbf{l} \succeq \mathbf{l}$.

Since $\succeq$ is a quasi-order, we know a partial ordering must result from the partition of $\mathcal{A}^+$ into a set of equivalence classes $\mathcal{A}^+_E$. In fact, the partitions are $\{[\top]\}, \{[\bot]\}, X_1, \ldots$ where $[\mathbf{l}]$ denotes all atoms that are alphabetic variants[28] of $\mathbf{l}$. That is, if $\mathbf{l}, \mathbf{m} \in X_i$ then there are substitutions μ and σ s.t. $\mathbf{l}\mu = \mathbf{m}$ and $\mathbf{m}\sigma = \mathbf{l}$. That is, $\succeq$ is a partial ordering over the set of equivalence classes of atoms $(\mathcal{A}^+_E)$. $(Mem(x_1, [x_1, y_1]), Mem(x_2, [x_2, y_2]) \ldots$ are examples of members of an equivalence class.)

Recall that the difference between subsumption and implication in first-order logic arose with the appearance of self-recursive clauses. Since there is no possibility of this with atoms in first-order logic, subsumption and implication are equivalent, and we can see that logical implication (*models*) over atoms is also a quasi-order over atoms.

[28] Two atoms are subsume-equivalent iff they are variants. This is not true for clauses in general.