Statistical Relational Learning Notes

Ganesh Ramakrishnan

September 4, 2008

Contents

Chapter 1

Sets, Relations and Logic

> 'Crime is common. Logic is rare. Therefore it is upon the logic rather than the crime that you should dwell.' Sherlock Holmes in Conan Doyle's *The Copper Breeches*.

1.1 Sets and Relations

1.1.1 Sets

A *set* is a fundamental concept in mathematics. Simply speaking, it consists of some objects, usually called its *elements*. Here are some basic notions about sets that you must already know about:

- A set S with elements a, b and c is usually written as $S = \{a, b, c\}$. The fact that a is an element of S is usually denoted by $a \in S$.
- A set with no elements is called the "empty set" and is denoted by $\emptyset$.
- Two sets S and T are equal ($S = T$) if and only if they contain precisely the same elements. Otherwise $S \neq T$.
- A set T is a subset of a set S ($T \subseteq S$) if and only if every element of T is also an element of S. If $T \subseteq S$ and $S \subseteq T$ then $S = T$. Sometimes $T \subseteq S$ may sometimes also be written as $S \supseteq T$. If $T \subseteq S$ and S has at least one element not in T, then $T \subset S$ (T is said to be "proper subset" of S). Again, $T \subset S$ may sometimes be written as $S \supset T$.

We now look at the meanings of the *union*, *intersection*, and *equivalence* of sets. The intersection, or product, of sets S and T, denoted by $S \cap T$ or ST or $S \cdot T$ consists of all elements common to both S and T. $ST \subset S$ and $ST \subseteq T$ for all sets S and T. Now, if S and T have no elements in common, then they are said to be *disjoint* and $ST = \emptyset$. It should be easy for you to see that $\emptyset \subseteq S$ for all S and $\emptyset \cdot S = \emptyset$ for all S. The union, or sum, of sets S and T, denoted

by $S \cup T$ or $S + T$, is the set consisting of elements that belong at least to S or T. Once again, it should be a straightforward matter to see $S \subseteq S + T$ and $T \subseteq S + T$ for all S and T. Also, $S + \emptyset = S$ for all S. Finally, if there is a one-to-one correspondence between the elements of set S and set T, then S and T are said to be equivalent ($S \sim T$). Equivalence and subsets form the basis of the definition of an infinte set: if $T \subset S$ and $S \sim T$ then S is said to be an infinite set. The set of natural numbers $\mathcal{N}$ is an example of an infinite set (any set $S \sim \mathcal{N}$ is said to be *countable* set).

1.1.2 Relations

A finite sequence is simply a set of n elements with a $1-1$ correspondence with the set $\{1, \ldots, n\}$ arranged in order of succession (an *ordered pair*, for example, is just a finite sequence with 2 elements). Finite sequences allow us to formalise the concept of a relation. If A and B are sets, then the set $A \times B$ is called the *cartesian product* of A and B and is denoted by all ordered pairs (a, b) such that $a \in A$ and $b \in B$. Any subset of $A \times B$ is a binary relation, and is called a relation from A to B. If $(a, b) \in R$, then aRb means "a is in relation R to b" or, "relation R holds for the ordered pair (a, b)" or "relation R holds between a and b." A special case arises from binary relations within elements of a single set (that is, subsets of $A \times A$). Such a relation is called a "relation in A" or a "relation over A". There are some important kinds of properties that may hold for a relation R in a set A:

Reflexive. The relation is said to be reflexive if the ordered pair $(a, a) \in R$ for every $a \in A$.

Symmetric. The relation is said to be symmetric if $(a, b) \in R$ *iff* $(b, a) \in R$ for $a, b \in A$.

Transitive. The relation is said to be transitive if $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$ for $a, b, c \in A$.

Here are some examples:

- The relation $\leq$ on the set of integers is reflexive and transitive, but not symmetric.
- The relation $R = \{(1,1), (1,2), (2,1), (2,2), (3,3), (4,4)\}$ on the set $A = \{1, 2, 3, 4\}$ is reflexive, symmetric and transitive.
- The relation $\div$ on the set $\mathcal{N}$ defined as the set $\{(x, y) : \exists z \in \mathcal{N} \text{ s.t. } xz = y\}$ is reflexive and transitive, but not symmetric.
- The relation $\perp$ on the set of lines in a plane is symmetric but neither reflexive nor transitive.

It should be easy to see a relation like R above is just a set of ordered pairs. Functions are just a special kind of binary relation F which is such that if $(a, b) \in F$ and $(a, c) \in F$ then $b = c$. Our familiar notion of a function F from a set A to a set B is one which associates with each $a \in A$ exactly one element $b \in B$ such that $(a, b) \in F$. Now, a function from a set A to itself is usually called a *unary operation* in A. In a similar manner, a *binary operation* in A is a function from $A \times A$ to A (recall $A \times A$ is the Cartesian product of A with itself: it is sometimes written as A^2). For example, if $A = \mathcal{N}$, then addition $(+)$ is a binary operation in A. In general, an n-ary operation F in A is a function from A^n to A, and if it is defined for every element of A^n, then A is said to be *closed* with respect to the operation F. A set which is closed for one or more n-ary operations is called an *algebra*, and a sub-algebra is a subset of such a set that remains closed with respect to those operations. For example:

- $\mathcal{N}$ is closed wrt the binary operations of $+$ and $\times$, and $\mathcal{N}$ along with $+, \times$ form an algebra.
- The set $\mathcal{E}$ of even numbers is a subalgebra of algebra of $\mathcal{N}$ with $+, \times$. The set $\mathcal{O}$ of odd numbers is not a subalgebra.
- Let $S \subseteq U$ and $S' \subseteq U$ be the set with elements of U not in S (the unary operation of complementation). Let $U = \{a, b, c, d\}$. The subsets of U with the operations of complementation, intersection and union form an algebra. (How many subalgebras are there of this algebra?)

Equivalence Relations

Any relation R in a set A for which all three properties hold (that is, R is reflexive, symmetric, and transitive) is said to be an "equivalence relation". Suppose, for example, we are looking at the relation R over the set of natural numbers $\mathcal{N}$, which consists of ordered pairs (a, b) such that $a + b$ is even[1] You should be able to verify that R is an equivalence relation over $\mathcal{N}$. In fact, R allows us to split $\mathcal{N}$ into two disjoint subsets: the set of odd numbers $\mathcal{O}$ and the set of even numbers $\mathcal{E}$ such that $\mathcal{N} = \mathcal{O} \cup \mathcal{E}$ and R is an equivalence relation over each of $\mathcal{O}$ and $\mathcal{E}$. This brings us to an important property of equivalence relations:

Theorem 1 *Any equivalence relation E over a set S partitions S into disjoint non-empty subsets $S_1, \ldots, S_k$ such that $S = S_1 \cup \cdots \cup S_k$.*

Let us see how E can be used to partition S by constructing subsets of S in the following way. For every $a \in S$, if $(a, b) \in E$ then a and b are put in the same subset. Let there be k such subsets. Now, since $(a, a) \in E$ for every $a \in S$, every element of S is in some subset. So, $S = S_1 \cup \cdots \cup S_k$. It also follows that the subsets are disjoint. Otherwise there must be some $c \in S_i, S_j$. Clearly, S_i

[1] Equivalence is often denoted by $\approx$. Thus, for an equivalence relation E, if $(a, b) \in E$, then $a \approx b$.

and S_j are not singleton sets. Suppose S_i contains at least a and c. Further let there be a $b \notin S_i$ but $b \in S_j$. Since $a, c \in S_i$, $(a, c) \in E$ and since $c, b \in S_j$, $(c, b) \in E$. Thus, we have $(a, c) \in E$ and $(c, b) \in E$, which must mean that $(a, b) \in E$ (E is transitive). But in this case b must be in the same subset as a by construction of the subsets, which contradicts our assumption that $b \notin S_i$. The converse of this is also true:

Theorem 2 *Any partition of a set S partitions into disjoint non-empty subsets $S_1, \ldots, S_k$ such that $S = S_1 \cup \cdots \cup S_k$ results in an equivalence relation over S.*

(Can you prove that this is the case? Start by constructing a relation E, with $(a, b) \in E$ if and only if a and b are in the same block, and prove that E is an equivalence relation.)

Each of the disjoint subsets $S_1, S_2, \ldots$ are called "equivalence classes", and we will denote the equivalence class of an element a in a set S by $[a]$. That is, for an equivalence relation E over a set S:

$$[a] = \{x : x \in S, (a, x) \in E\}$$

What we are saying above is that the collection of all equivalence classes of elements of S forms a partition of S; and conversely, given a partition of the set S, there is an equivalence relation E on S such that the sets in the partition (sometimes also called its "blocks") are the equivalence classes of S.

Partial Orders

Given an equality relation $=$ over elements of a set S, a partial order $\preceq$ over S is a relation over S that satisfies the following properties:

Reflexive. For every $a \in S$, $a \preceq a$

Anti-Symmetric. If $a \preceq b$ and $b \preceq a$ then $a = b$

Transitive. If $a \preceq b$ and $b \preceq c$ then $a \preceq c$

Here are some properties about partial orders that you should know (you will be able to understand them immediately if you take, as a special case, $\preceq$ as meaning $\leq$ and $\prec$ as meaning $<$):

- If $a \preceq b$ and $a \neq b$ then $a \prec b$
- $b \succeq a$ means $a \preceq b$, $b \succ a$ means $a \prec b$
- If $a \preceq b$ or $b \preceq a$ then a, b are comparable, otherwise they are not comparable.

A set S over which a relation of partial order is defined is called a *partially ordered set*. It is sometimes convenient to refer to a set S and a relation R defined over S together by the pair $< S, R >$. So, here are some examples of partially ordered sets $< S, \preceq >$:

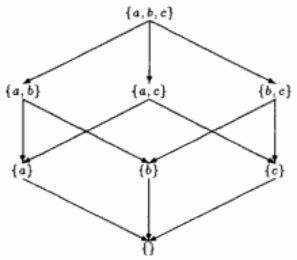


Figure 1.1: The lattice structure of $\langle S, \preceq \rangle$, where S is the power set of $\{a, b, c\}$.

- S is a set of sets, $S_1 \preceq S_2$ means $S_1 \subseteq S_2$
- $S = \mathcal{N}$, $n_1 \preceq n_2$ means $n_1 = n_2$ or there is a $n_3 \in \mathcal{N}$ such that $n_1 + n_3 = n_2$
- S is the set of equivalence relations $E_1, \ldots$ over some set T, $E_L \preceq E_M$ means for $u, v \in T$, uE_Lv means uE_Mv (that is, $(u, v) \in E_L$ means $(u, v) \in E_M$).

Given a set $S = \{a, b, \ldots\}$ if $a \prec b$ and there is no $x \in S$ such that $a \prec x \prec b$ then we will say b *covers* a or that a is a *downward cover* of b. Now, suppose S_{down} be a set of downward covers of $b \in S$. If for all $x \in S$, $x \prec b$ implies there is an $a \in S_{down}$ s.t. $x \preceq a \prec b$, then S_{down} is said to be a *complete* set of downward covers of b. Partially ordered sets are usually shown as diagrams like in Figure 1.1.
The diagrams, as you can see, are graphs (sometimes called Hasse graphs or Hasse diagrams). In the graph, vertices represent elements of the partially ordered set. A vertex v_2 is at a higher level than vertex v_1 whenever $v_1 \prec v_2$, and there is an edge between the two vertices only if v_2 covers v_1 (that is, v_2 is an immediate predecessor). The graph is therefore really a directed one, in which there is a directed edge from a vertex v_2 to v_1 whenever v_2 covers v_1. Also, since the relation is anti-symmetric, there can be no cycles. So, the graph is a directed acyclic graph, or DAG.

In the diagram in Figure **??** on the left, S is the set of non-empty subsets of $\{a, b, c\}$ and $\preceq$ denotes the subset relationship (that is, $S_1 \preceq S_2$ if and only if $S_1 \subset S_2$). The diagram on the right is an example of a *chain*, or a *totally ordered* set.

You should be able to see that a finite chain of length n can be put in a one-to-one correspondence to a finite sequence of natural numbers $(1, \ldots, n)$ (the correct way to say this is that a finite chain is isomorphic with a finite sequence of natural numbers). In general, a partially ordered set S is a chain if for every pair $a, b \in S$, $a \prec b$ or $b \prec a$. There is a close relationship between a partially ordered set and a chain. Suppose S is a partially ordered set. We

can always associate a function f from the elements of S to $\mathcal{N}$ (the set of natural numbers), so that if $a \prec b$ for $a, b \in S$, then $f(a) < f(b)$. f is called a *consistent enumeration* of S, and is not unique and we can use it to define a chain consistent with S. (We will leave the proof of the existence a consistent enumeration for you. One way would be to use the method of induction on the number of elements in S: clearly there is such an enumeration for $|S| = 1$. Assume that an enumeration exists for $|S| = n - 1$ and prove it for $|S| = n$.)

Some elements of a partially ordered set have special properties. Let $< S, \preceq >$ be a p.o. set and $T \subseteq S$. Then (in the following, you should read the symbol $\exists$ as being shorthand for "there exists", and $\forall$ as "for all"):

– Least element of T $a \in T$ *s.t.* $\forall t \in T\ a \preceq t$	– Greatest element of T $a \in T$*s.t.* $\forall t \in T\ a \succeq t$
– Least element, if it exists, is unique. If $T = S$ this is the "zero" element	– Greatest element, if it exists is unique. If $T = S$ then this is the "unity" element
– Minimal element of T $a \in T \quad \nexists t \in T$ *s.t.* $t \prec a$	– Maximal element of T $a \in T \quad \nexists t \in T$ *s.t.* $t \succ a$
– Minimal element need not be unique	– Maximal element need not be unique
– Lower bound of T $b \in S$ *s.t.* $b \preceq t\ \forall t \in T$	– Upper bound of T $b \in S$ *s.t.* $b \succeq t\ \forall t \in T$
– Glb g of T $b \preceq g\ \forall b, g :$ *lbs of* T	– Lub g of T $b \succeq g\ \forall b, g :$ *ubs of* T
– If it exists, the glb is unique	– If it exists the lub is unique

As you would have observed, there is a difference between a least element and a minimal element (and correspondingly, between greatest and maximal elements). The requirement of a minimal (maximal) upper bound is, in some sense, a weakening of the requirement of a least (greatest) upper bound. If x and y are both lub's of some set $T \subseteq S$, then $y \preceq x$ and $z \preceq y$, so then $x \approx y$. This means that all lub's of T are equivalent. Dually, if x and y are glb's of some T, then also $x \approx y$. Thus, if a least element exists, then it is unique: this is not necessarily the case with a minimal element. Also, least and greatest elements must belong to the set T, but lower and upper bounds need not.
For this example, S has: (1) one upper bound b; (2) no lower bound; (3) a greatest element b; (4) no least element; (5) no greatest lower bound; (6) two minimal elements a and e; and (7) one maximal element b. Can you identify what the corresponding statements are for T?

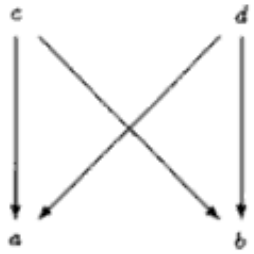


Figure 1.2: $\{a, b\}$ has no lub here.

The glb and lub are sometimes also taken to be binary operations on a partially ordered set S, that assigns to an ordered pair in S^2 the corresponding glb or lub. The first operation is called the *product* or *meet* and is denoted by $\cdot$ or $\sqcap$. The second operation is sometimes called the *sum* or *join* and is denoted by $+$ or $\sqcup$.

In a quasi-ordered set, a subset need not have a lub or glb. We will take an example to illustrate this. Let $S = \{a, b, c, d\}$, and let $\preceq$ be defined as $a \preceq c$, $b \preceq c$, $a \preceq d$ and $d \preceq b$. Then since c and d are incomparable, the set a, b has no lub in this quasi-order. See Figure 1.2.

Similarly, a set need not have a maximal or a minimal, nor upward or downward covers. For instance, let S be the infinite set $\{y, x_l, x_2, x_3, \ldots\}$, and let $\preceq$ be a quasi-order on S, defined as $y \prec \ldots x_{n+1} \prec x_n \prec \ldots \prec x_2 \prec x_1$. Then there is no upward cover of y: for every x_n, there always is an x_{n+l} such that $y \prec x_{n+1} \prec x_n$. In this case, y has no complete set of upward covers.

Note that a complete set of upward covers for y need not contain all upward covers of y. However, in order to be complete, it should contain at least one element from each equivalence class of upward covers. On the other hand, even the set of all upward covers of y need not be complete for y. For the example given above, the set of all upward covers of y is empty, but obviously not complete.

A notion of some relevance later is that of a function f defined on a partially ordered set $< S, \preceq >$. Specifically, we would like to know if the function is: (a) monotonic; and (b) continuous. Monotonicity first:

> A function f on $< S, \preceq >$ is monotonic if and only if for all $u, v \in S$, $u \preceq v$ means $f(u) \preceq f(v)$

Now, suppose a subset S_1 of S have a least upper bound $lub(S_1)$ (with some abuse of notation: here $lub(X)$ is taken to be the lub of the elements in set X). Such subsets are called "directed" subsets of S. Then:

> A function f on $< S, \preceq >$ is continuous if and only if for all directed subsets S_i of S, $f(lub(S_i)) = lub(\{f(x) : x \in S_i\})$.

That is, if a directed set S_i has a least upper bound $lub(S_i)$, then the set obtained by applying a continuous function f to the elements of S_i has least upper bound $f(lub(S_i))$. Functions that are both monotonic and continuous on some partially ordered set $< S, \preceq >$ are of interest to us because they can be used, for some kinds of orderings, to guarantee that for some $s \in S$, $f(s) = s$. That is, f is said to have a "fixpoint".

Lattices

A lattice is just a partially ordered set $< S, \preceq >$ in which every pair of elements $a, b \in S$ has a glb (represented by $\sqcap$) and a lub (represented by $\sqcup$). From the definitions of lower and upper bounds, we are able to show that in any such partially ordered set, the operations will have the following properties:

- $a \sqcap b = b \sqcap a$, and $a \sqcup b = b \sqcup a$ (that is, they are are commutative).
- $a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c$, and $a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$ (that is, they are associative).
- $a \sqcap (a \sqcup b) = a$, and $a \sqcup (a \sqcap b) = a$ (that is, they are "absorptive").
- $a \sqcap b = a$ and $a \sqcup b = b$.

We will not go into all the proofs here, but show one for illustration. Since $a \sqcap b$ is the glb of a and b, $a \sqcap b \preceq a$. Clearly then $a \sqcup (a \sqcap b)$, which is the lub of a and $a \sqcap b$, is a. This is one of the absorptive properties above. You should also be able to see, from these properties, that a lattice can also be seen simply as an algebra with two binary operations $\sqcap$ and $\sqcup$ that are commutative, associative and absorptive.

Theorem 3 *A lattice is an algebra with the binary operations of* $\sqcup$ *and* $\sqcap$.

Here is an example of a lattice: let S be all the subsets of $\{a, b, c\}$, and for $X, Y \in S$, $X \preceq Y$ means $X \subseteq Y$, $X \sqcap Y = X \cap Y$ and $X \sqcup Y = X \cup Y$. Then $< S, \subseteq >$ is a lattice. The empty set $\emptyset$ is the zero element, and S is the unity element of the lattice. More generally, a lattice that has a zero or least element (which we will denote $\bot$), and a unity or greatest element (which we will denote $\top$) is called a *bounded* lattice. In such lattices, the following necessarily hold: $a \sqcup \top = \top$; $a \sqcap \top = a$; $a \sqcup \bot = a$; and $a \sqcap \bot = \bot$. A little thought should convince you that a finite lattice will always be bounded: if the lattice is the set $S = \{a_1, \ldots, a_n\}$ then $\top = a_1 \sqcup \cdots \sqcup a_n$ and $\bot = a_1 \sqcap \cdots \sqcap a_n$. (But, does the reverse hold: will a bounded lattice always be finite?)

Two properties of subsets of lattices are of interest to us. First, a subset M of a lattice L is called a *sublattice* of L if M is also closed under the same binary operations of $\sqcup$ and $\sqcap$ defined for L (that is, M is a lattice with the same operations as those of L). Second, if a lattice L has the property that every subset of L has a lub and a glb, then the L is said to be a *complete* lattice. Clearly, every finite lattice is complete. Further, since every subset of

L has a lub and a glb, this must certainly be true of L itself. So, L has a lub, which must necessarily be the greatest element of L. Similarly, L has a glb, which must necessarily be the least element of L. In fact, the elements of L are ordered in such a way that each element is on some path from $\top$ to $\bot$ in the Hasse diagram. An example of an ordered set that is always a complete lattice is the set of all subsets of a set S, ordered by $\subseteq$, with binary operations $\cap$ and $\cup$ for the glb and lub. This set, the "powerset" of S, is often denoted by 2^S. So, if $S = \{a, b, c\}$, 2^S is the set $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$. Clearly, every subset of s of 2^S has both a glb and a lub in S.

There are two important results concerning complete lattices and functions defined on them. The Knaster-Tarski Theorem tells us that every monotonic function on a complete lattice $< S, \preceq >$ has a least fixpoint.

Theorem 4 *Let $< S, \preceq >$ be a complete lattice and let $f : S \rightarrow S$ be a monotonic function. Then the set of fixed points of f in L is also a complete lattice $< P, \preceq >$ (which obviously means that f has a greatest as well as a least fixpoint).*

Proof Sketch:[2] Let $D = \{x | x \preceq f(x)\}$. From the very definition of D, it follows that every fixpoint is in D. Consider some $x \in D$. Then because f is monotone we have $f(x) \preceq f(f(x))$. Thus,

$$\forall\ x \in D,\ f(x) \in D \tag{1.1}$$

Let $u = lub(D)$ (which should exist according to our assumption that $< S, \preceq >$ is a complete lattice. Then $x \preceq u$ and $f(x) \preceq f(u)$, so $x \preceq f(x) \preceq f(u)$. Therefore $f(u)$ is an upper bound of D. However, u is the least upper bound, hence $u \preceq f(u)$, which in turn implies that, $u \in D$. From (1.1), it follows that $f(u) \in D$. From $u = lub(D)$, $f(u) \in D$ and $u \preceq f(u)$, it follows that $f(u) = u$. Because every fixpoint is in D we have that u is the greatest fixpoint of f. Similarly, it can be proved that if $E = \{x | f(x) \preceq x\}$, then $v = glb(E)$ is a fixed point and therefore the smallest fixpoint of f. □

Kleene's First Recursion Theorem tells us how to find the element $s \in S$ that is the least fixpoint, by incrementally constructing lubs starting from applying a continuous function to the least element of the lattice ($\bot$).

Theorem 5 *Let S be a complete partial order and let $f : S \rightarrow S$ be a continuous (and therefore monotone) function. Then the least fixed point of f is the supremum of the ascending Kleene chain of f:*

$$\bot \preceq f(\bot) \preceq f(f(\bot)) \preceq \ldots \preceq f^n(\bot) \preceq \ldots$$

In the special case that $\preceq$ is $\subseteq$, the incremental procedure starts with the empty set $\emptyset$, and progressive lub's are obtained by application of the set-union operation $\cup$. We will not give the proofs of this result here.

[2] Can you complete the proof?

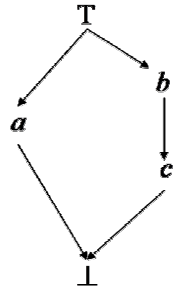


Figure 1.3: Example lattice for illustrating the concept of lattice length.

A final concept we will need is the concept of the length of a lattice. For a pair of elements a, b in a lattice L such that $a \preceq b$, the interval $[a, b]$ is the set $\{x : x \in L, \quad a \preceq x \preceq b\}$. Now, consider a subset of $[a, b]$ that contains both a and b, and is such that any pair of elements in the subset are comparable. Then that subset is a chain from a to b: if the number of elements in the subset is n, then the length of the chain is $n - 1$. Maximal chains from a to b are those of the form $a = x_1 \prec x_2 \prec \cdots \prec x_n = b$ such that each x_i is covered by x_{i+1}. If all maximal chains from a to b are finite, then the longest of these defines the length of the interval $[a, b]$. For a bounded lattice, the length of the interval $[\bot, \top]$ defines the length of the lattice. So, in the lattice in Figure 1.3, there are two maximal chains between $\bot$ and $\top$, of lengths 2 and 3 (what are these?). The length of lattice is thus equal to 3. Now, it should be evident that finite lattices will always have a finite length, but it is possible for lattices to have a finite length, but have infinitely many elements. For example, the lattice $L = \{\bot, \top, x_1, x_2, \ldots\}$ such that $\bot \prec x_i \prec \top$ has a finite length (all maximal chains are of length 2). (Indeed, it is even possible to have an infinite set in which maximal chains are of finite, but increasing in lengths of $1, 2 \ldots$.)

Quasi-Orders

A quasi-order Q in a set S is a binary relation over S that satisfies the following properties:

Reflexive. For every $a \in S$, aQa

Transitive. If aQb and bQc then aQc

You can see that a quasi-order differs from an equivalence relation in that symmetry is not required. Further, it differs from a partial order because no equality is defined, and therefore the property of anti-symmetry property cannot be defined either. There are two important properties of quasi-orders, which we will present here without proof:

- If a quasi-order Q is defined on a set $S = \{a, b, \ldots\}$, and we define a binary relation E as follows: aEb iff aQb and bQa. Then E is an equivalence relation.
- Let E partition S into subsets $X, Y, \ldots$ of equivalent elements. Let $T = \{X, Y, \ldots\}$ and $\preceq$ be a binary relation in T meaning $X \preceq Y$ in T if and only if xQy in S for some $x \in X, y \in Y$. Then T is partially ordered by $\preceq$.

What these two properties say is simply this:

> A quasi-order order Q over a set S results in a partial ordering over a set of equivalence classes of elements in S.

1.2 Logic

Logic, the study of arguments and 'correct reasoning', has been with us for at least the better part of two thousand years. In Greece, we associate its origins with Aristotle (384 B.C.–322 B.C.); in India with Gautama and the Nyaya school of philosophy (3rd Century B.C.?); and in China with Mo Ti (479 B.C.–381 B.C.) who started the Mohist school. Most of this dealt with the use and manipulation of *syllogisms*. It would only be a small injustice to say that little progress was made until Gottfried Wilhelm von Leibniz (1646–1716). He made a significant advance in the use of logic in mathematics by introducing symbols to represent statements and relations. Leibniz hoped to reduce all errors in human reasoning to minor calculational mistakes. Later, George Boole (1815–1864) established the connection between logic and sets, forming the basis of *Boolean algebra*. This link was developed further by John Venn (1834–1923) and Augustus de Morgan (1806–1872). It was around this time that Charles Dodgson (1832–1898), writing under the pseudonym Lewis Carroll, wrote a number of popular logic textbooks. Fundamental changes in logic were brought about by Friedrich Ludwig Gottlob Frege (1848–1925), who strongly rejected the idea that the laws of logic are synonymous with the laws of thought. For Frege, the former were laws of *truth*, having little to say on the processes by which human beings represent and reason with reality. Frege developed a logical framework that incorporated propositions with relations and the validity of arguments depended on the relations involved. Frege also introduced the device of *quantifiers* and *bound variables*, thus laying the basis for *predicate logic*, which forms the basis of all modern logical systems. All this and more is described by Bertrand Russell (1872–1970) and Alfred North Whitehead (1861–1947) in their monumental work, *Principia Mathematica*. And then in 1931, Kurt Gödel (1906–1978) showed much to the dismay of mathematicians everywhere that formal systems of arithmetic would remain incomplete.

Rational agents require knowledge of their world in order to make rational decisions. With the help of a declarative (knowledge representation) language, this knowledge (or a portion of it) is represented and stored in a knowledge

base. A knowledge-base is composed of sentences in a language with a truth theory (logic), so that someone external to the system can interpret sentences as statements about the world (semantics). Thus, to express knowledge, we need a precise, *declarative* language. By a *declarative language*, we mean that

1. The system believes a statement S *iff* it considers S to be true, since one cannot believe S without an idea of what it means for the world to fulfill S.
2. The knowledge-based must be precise enough so that we must know, (1) which symbols represent sentences, (2) what it means for a sentence to be true, and (3) when a sentence follows from other sentences.

Two declarative languages will be discussed in this chapter: (0 order or) propositional logic and first order logic.

1.3 Propositional Logic

Formal logic is concerned with statements of fact, as opposed to opinions, commands, questions, exclamations *etc.* Statements of fact are assertions that are either true or false, the simplest form of which are called *propositions*. Here are some examples of propositions:

The earth is flat.
Humans are monkeys.
$1 + 1 = 2$

At this stage, we are not saying anything about whether these are true or false: just that they are sentences that are one or the other. Here are some examples of sentences that are not propositions:

Who goes there?
Eat your broccoli.
This statement is false.

It is normal to represent propositions by letters like $P, Q, \ldots$. For example, P could represent the proposition 'Humans are monkeys.' Often, simple statements of fact are insufficient to express complex ideas. *Compound statements* can be combining two or more propositions with *logical connectives* (or simply, connectives). The connectives we will look at here will allow us to form sentences like the following:

It is not the case that P
P and Q
P or Q
P if Q

The P's and Q's above are propositions, and the words underlined are the connectives. They have special symbols and names when written formally:

Statement	Formally	Name
It is not the case that P	$\neg P$	Negation
P and Q	$P \wedge Q$	Conjunction
P or Q	$P \vee Q$	Disjunction
P if Q	$P \leftarrow Q$	Conditional

There is, for example, a form of argument known to logicians as the *disjunctive syllogism*. Here is one due to the Stoic philosopher Chrysippus, about a dog chasing a rabbit. The dog arrives at a fork in the road, sniffs at one path and then dashes down the other. Chrysippus used formal logic to describe this:[3]

Statement	Formally
The rabbit either went down Path A or Path B.	$P \vee Q$
It did not go down Path A.	$\neg P$
Therefore it went down Path B.	$\therefore Q$

Here P represents the proposition 'The rabbit went down Path A' and Q the proposition 'The rabbit went down Path B.' To argue like Chrysippus requires us to know how to write correct logical sentences, ascribe truth or falsity to propositions, and use these to derive valid consequences. We will look at all these aspects in the sections that follow.

1.3.1 Syntax

Every language needs a *vocabulary*. For the language of propositional logic, we will restrict the vocabulary to the following:

Propositional symbols:	$P, Q, \ldots$
Logical connectives[4] **:**	$\neg, \wedge, \vee, \leftarrow$
Brackets:	$(,)$

The next step is to specify the rules that decide how legal sentences are to be formed within the language. For propositional logic, legal sentences or *well-formed formulæ* (wffs for short) are formed using the following rules:

1. Any propositional symbol is a wff;
2. If α is a wff then $\neg\alpha$ is a wff; and

[3] There is no suggestion that the principal agent in the anecdote employed similar means of reasoning.

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

Wffs consisting simply of propositional symbols (Rule 1) are sometimes called *atomic* wffs and others *compound* wffs Informally, it is acceptable to drop outermost brackets. Here are some examples of wffs and 'non-wffs':

Formula	Comment
$(\neg P)$	Not a wff. Parentheses are only allowed with the connectives in Rule 3
$\neg\neg P$	P is wff (Rule 1), $\neg P$ is wff (Rule 2), $\therefore \neg\neg P$ is wff (Rule 2)
$(P \leftarrow (Q \wedge R))$	P, Q, R are wffs (Rule 1), $\therefore (Q \wedge R)$ is a wff (Rule 3), $\therefore (P \leftarrow (Q \wedge R))$ is a wff (Rule 3)
$P \leftarrow (Q \wedge R)$	Not a wff, but acceptable informally
$((P) \wedge (Q))$	Not a wff. Parentheses are only allowed with the connectives in Rule 3
$(P \wedge Q \wedge R)$	Not a wff. Rule 3 only allows two symbols within a pair of brackets

One further kind of informal notation is widespread and quite readable. The conditional $(P \leftarrow ((Q_1 \wedge Q_2) \dots Q_n)))$ is often written as $(P \leftarrow Q_1, Q_2, \dots Q_n)$ or even $P \leftarrow Q_1, Q_2, \dots Q_n$.

It is one thing to be able to write legal sentences, and quite another matter to be able to assess their truth or falsity. This latter requires a knowledge of semantics, which we shall look at shortly.

Normal Forms

Every formulae in propositional logic is equivalent to a formula that can be written as a conjunction of disjunctions. That is, something like $(A \vee B) \wedge (C \vee D) \wedge \cdots$. When written in this way the formula is said to be in *conjunctive normal form* or CNF. There is another form, which consists of a disjunction of conjunctions, like $(A \wedge B) \vee (C \wedge D) \vee \cdots$, called the *disjunctive normal form* or DNF. In general, a formula F in CNF can be written somewhat more cryptically as:

$$F = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m} L_{i,j} \right)$$

and a formula G in DNF as:

$$G = \bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{m} L_{i,j} \right)$$

Here, $\bigvee F_i$ is short for $F_1 \vee F_2 \vee \cdots$ and $\bigwedge F_i$ is short for $F_i \wedge F_2 \wedge \cdots$. In both CNF and DNF forms above, the $L_{i,j}$ are either propositions or negations of propositions (we shall shortly call these "literals").

1.3.2 Semantics

There are three important concepts to be understood in the study of semantics of well-formed formulæ: *interpretations*, *models*, and *logical consequence*.

Interpretations

For propositional logic, an *interpretation* is simply an assignment of either *true* or *false* to all propositional symbols in the formula. For example, given the wff $(P \leftarrow (Q \wedge R))$ here are two different interpretations:

	P	Q	R
I_1 :	true	false	true
I_2 :	false	true	true

You can think of I_1 and I_2 as representing two different 'worlds' or 'contexts'. After a moment's thought, it should be evident that for a formula with N propositional symbols, there can never be more than 2^N possible interpretations.

Truth or falsity of a wff only makes sense given an interpretation (by the principle of bivalence, any interpretation can only result in a wff being either *true* or *false*). Clearly, if the wff simply consists of a single propositional symbol (recall that this was called an atomic wff), then the truth-value is simply that given by the interpretation. Thus, the wff P is *true* in interpretation I_1 and *false* in interpretation I_2. To obtain the truth-value of compound wffs like $(P \leftarrow (Q \wedge R))$ requires a knowledge the semantics of the connectives. These are usually summarised in a tabular form known as *truth tables*. The truth tables for the connectives of interest to us are given below.

Negation. Let α be a wff[5]. Then the truth table for $\neg\alpha$ is as follows:

α	$\neg\alpha$
false	true
true	false

Conjunction. Let α and β be wffs. The truth table for $(\alpha \wedge \beta)$ is as follows:

[5]We will use Greek characters like α, β to stand generically for any wff.

α	β	$(\alpha \wedge \beta)$
false	false	false
false	true	false
true	false	false
true	true	true

Disjunction. Let α and β be wffs. The truth table for $(\alpha \vee \beta)$ is as follows:

α	β	$(\alpha \vee \beta)$
false	false	false
false	true	true
true	false	true
true	true	true

Conditional. Let α and β be wffs. The truth table for $(\alpha \leftarrow \beta)$ is as follows:

α	β	$(\alpha \leftarrow \beta)$
false	false	true
false	true	false
true	false	true
true	true	true

We are now in a position to obtain the truth-value of a compound wff. The procedure is straightforward: given an interpretation, we find the truth-values of the smallest 'sub-wffs' and then use the truth tables for the connectives to obtain truth-values for increasingly complex sub-wffs. For $(P \leftarrow (Q \wedge R))$ this means:

1. First, obtain the truth-values of P, Q, R using the interpretation;
2. Next, obtain the truth-value of $(Q \wedge R)$ using the truth table for 'Conjunction' and the truth-values of Q and R (Step 1); and
3. Finally, obtain the truth-value $(P \leftarrow (Q \wedge R))$ using the truth table for 'Conditional' and the truth-values of P (Step 1) and $(Q \wedge R)$ (Step 2).

For the interpretations I_1 and I_2 earlier these truth-values are as follows:

	P	Q	R	$(Q \wedge R)$	$(P \leftarrow (Q \wedge R))$
I_1 :	true	false	true	false	true
I_2 :	false	true	true	true	false

Thus, $(P \leftarrow (Q \wedge R))$ is *true* in interpretation I_1 and *false* in I_2.

Models

Every interpretation (that is, an assignment of truth-values to propositional symbols) that makes a well-formed formula *true* is said to be a *model* for that formula. Take for example, the two interpretations I_1 and I_2 above. We have already seen that I_1 is a model for $(P \leftarrow (Q \wedge R))$; and that I_2 is not a model for the same formula. In fact, I_1 is also a model for several other wffs like: P, $(P \wedge R)$, $(Q \vee R)$, $(P \leftarrow Q)$, *etc.* Similarly, I_2 is a model for Q, $(Q \wedge R)$, $(P \vee Q)$, $(Q \leftarrow P)$, *etc.*

As another example, let $\{P, Q, R\}$ be the set of all atoms in the language, and α be the formula $((P \wedge Q) \leftrightarrow (R \rightarrow Q))$. Let I be the interpretation that makes P and R true, and Q false (so $I = \{P, R\}$). We determine whether α is true or false under I as follows:

1. P is true under I, and Q is false under I, so $(P \wedge Q)$ is false under I.
2. R is true under I, Q is false under I, so $(R \rightarrow Q)$ is false under I.
3. $(P \wedge Q)$ and $(R \rightarrow Q)$ are both false under I, so α is true under I.

Since α is true under I, I is a model of α. Let $I' = \{P\}$. Then $(P \wedge Q)$ is false, and $(R \rightarrow Q)$ is true under I'. Thus α is false under I', and I' is not a model of α.

The definition of model can be extended to a set of formulæ; an interpretation I is said to be a model of a set of formulæ Σ if I is a model of all formulæ $\alpha \in \Sigma$. Σ is then said to have I as a model. We will offer an example to illustrate this extended definition. Let $\Sigma = \{P, (Q \vee I), (Q \rightarrow R)\}$, and let $I = \{P, R\}$, $I' = \{P, Q, R\}$, and I" $= \{P, Q\}$ be interpretations. I and I' satisfy all formulas in Σ, so I and I' are models of Σ. On the other hand, I" falsifies $(Q \rightarrow R)$, so I" is not a model of Σ.

At this point, we can distinguish amongst two kinds of formulæ:

1. A wff may be such that *every* interpretation is a model. An example is $(P \vee \neg P)$. Since there is only one propostional symbol involved (P), there are at most $2^1 = 2$ interpretations possible. The truth table summarising the truth-values for this formula is:

	P	$\neg P$	$(P \vee \neg P)$
I_1 :	false	true	true
I_2 :	true	false	true

   $(P \vee \neg P)$ is thus *true* in every possible 'context'. Formulæ like these, for which every interpretation is a model are called *valid* or *tautologies*

2. A wff may be such that *none* of the interpretations is a model. An example is $(P \wedge \neg P)$. Again there is only one propostional symbol involved (P), and thus only two interpretations possible. The truth table summarising the truth-values for this formula is:

	P	$\neg P$	$(P \wedge \neg P)$
I_1 :	false	true	false
I_2 :	true	false	false

$(P \wedge \neg P)$ is thus *false* in every possible 'context'. Formulæ like these, for which none of the interpretations is a model are called *unsatisfiable* or *inconsistent*

Finally, any wff that has at least *one* interpretation as a model is said to be *satisfiable*.

Logical Consequence

We are often interested in establishing the truth-value of a formula given that of some others. Recall the Chrysippus argument:

Statement	Formally
The rabbit either went down Path A or Path B.	$P \vee Q$
It did not go down Path A.	$\neg P$
Therefore it went down Path B.	$\therefore Q$

Here, we want to establish that if the first two statements are true, then the third follows. The formal notion underlying all this is that of *logical consequence*. In particular, what we are trying to establish is that some well-formed formula α is the *logical consequence* of a conjunction of other well-formed formulæ Σ (or, that Σ *logically implies* α). This relationship is usually written thus:

$$\Sigma \models \alpha$$

Σ being the conjunction of several wffs, it is itself a well-formed formula[6]. Logical consequence can therefore also be written as the following relationship between a pair of wffs:

$$((\beta_1 \wedge \beta_2) \ldots \beta_n) \models \alpha$$

It is sometimes convenient to write Σ as the set $\{\beta_1, \beta_2, \ldots, \beta_n\}$ which is understood to stand for the conjunctive formula above. But how do we determine if this relationship between Σ and α does indeed hold? What we want is the following: whenever the statements in Σ are true, α must also be true. In formal terms, this means: $\Sigma \models \alpha$ *if every model of* Σ *is also model of* α. Decoded:

[6]There is therefore nothing special needed to extend the concepts of validity and unsatisfiability to conjunctions of formulæ like Σ. Thus, Σ is valid if and only if every interpretation is a model of the conjunctive wff (in other words, a model for each wff in the conjunction); and it is unsatisfiable if and only if none of the interpretations is a model of the conjunctive wff. It should be apparent after some reflection that if Σ is valid, then all logical consequences of it are also valid. On the other hand, if Σ is unsatisfiable, then any well-formed formula is a logical consequence.

- Recall that a model for a formula is an interpretation (assignment of truth-values to propositions) that makes that formula *true*;
- Therefore, a model for Σ is an interpretation that makes $((\beta_1 \wedge \beta_2) \ldots \beta_n)$ *true*. Clearly, such an interpretation will make each of $\beta_1, \beta_2, \ldots, \beta_n$ *true*;
- Let $I_1, I_2, \ldots, I_k$ be all the interpretations that satisfy the requirement above: that is, each is a model for Σ and there are no other models for Σ (recall that if there are N propositional symbols in Σ and α together, then there can be no more than 2^N such interpretations);
- Then to establish $\Sigma \models \alpha$, we have to check that each of $I_1, I_2, \ldots, I_k$ is also a model for α (that is, each of them make α *true*).

The definition of logical entailment can be extended to the entailment of sets of formulæ. Let Σ and Γ be sets of formulas. Then Γ is said to be a logical consequence of Σ (written as $\Sigma \models \Gamma$), if $\Sigma \models \alpha$, for every formula $\alpha \in \Gamma$. We also say Σ (logically) implies Γ.

We are now in a position to see if Chrysippus was correct. We wish to see if $((P \vee Q) \wedge \neg P) \models Q$. From the truth tables on page 17, we can construct a truth table for $((P \vee Q) \wedge \neg P)$:

	P	Q	$(P \vee Q)$	$\neg P$	$((P \vee Q) \wedge \neg P)$
I_1 :	false	false	false	true	false
I_2 :	false	true	true	true	true
I_3 :	true	false	true	false	false
I_4 :	true	true	true	false	false

It is evident that of the four interpretations possible only one is a model for $((P \vee Q) \wedge \neg P)$, namely: I_2. Clearly I_2 is also a model for Q. Therefore, every model for $((P \vee Q) \wedge \neg P)$ is also a model for Q[7]. It is therefore indeed true that $((P \vee Q) \wedge \neg P) \models Q$. In fact, you will find you can 'move' formulæ from left to right in a particular manner. Thus if:

$$((P \vee Q) \wedge \neg P) \models Q$$

then the following also hold:

$$(P \vee Q) \models (Q \leftarrow \neg P) \quad \text{and} \quad \neg P \models (Q \leftarrow (P \vee Q))$$

These are consequences of a a more general result known as the *deduction theorem*, which we look at now. Using a set-based notation, let $\Sigma = \{\beta_1, \beta_2, \ldots, \beta_i, \ldots, \beta_n\}$. Then, the deduction theorem states:

[7] Although I_4 is also a model for Q, the test for logical consequence only requires us to examine those interpretations that are models of $((P \vee Q) \wedge \neg P)$. This precludes I_4.

Theorem 6

$$\Sigma \models \alpha \text{ if and only if } \Sigma - \{\beta_i\} \models (\alpha \leftarrow \beta_i)$$

.

Proof: Consider first the case that $\Sigma \models \alpha$. That is, every model of Σ is a model of α. Now assume $\Sigma - \{\beta_i\} \not\models (\alpha \leftarrow \beta_i)$. That is, there is some model, say M, of $\Sigma - \{\beta_i\}$ that is not a model of $(\alpha \leftarrow \beta_i)$. That is, β_i is true and α is false in M. That is M is a model for $\Sigma - \{\beta_i\}$ and for β_i, but not a model for α. In other words, M is a model for Σ but not a model for α which is not possible. Therefore, if $\Sigma \models \alpha$ then $\Sigma - \{\beta_i\} \not\models (\alpha \leftarrow \beta_i)$. Now for the "only if" part. That is, let $\Sigma - \{\beta_i\} \models (\alpha \leftarrow \beta_i)$. We want to show that $\Sigma \models \alpha$. Once again, let us assume the contrary (that is, $\Sigma \not\models \alpha$. This means there must be a model M for Σ that is not a model for α. However, since $\Sigma = \{\beta_1, \beta_2, \ldots, \beta_i, \ldots, \beta_n\}$, M is both a model for $\Sigma - \{\beta_i\}$ and a model for each of the β_i. So, M cannot be a model for $(\alpha \leftarrow \beta_i)$. We are therefore in a position that there is a model M for $\Sigma - \{\beta_i\}$ that is not a model of $(\alpha \leftarrow \beta_i)$, which contradicts what was given. □

The deduction theorem isn't restricted to propositional logic, and holds for first-order logic as well. It can be invoked repeatedly. Here is an example of using it twice:

$$\Sigma \models \alpha \text{ if and only if } \Sigma - \{\beta_i, \beta_j\} \models (\alpha \leftarrow (\beta_i \wedge \beta_j))$$

With Chrysippus, applying the deduction theorem twice results in:

$$\{(P \vee Q), \neg P\} \models Q \text{ if and only if } \emptyset \models (Q \leftarrow ((P \vee Q) \wedge \neg P))$$

If $\emptyset \models (Q \leftarrow ((P \vee Q) \wedge \neg P))$ then every model for $\emptyset$ must be a model for $(Q \leftarrow ((P \vee Q) \wedge \neg P))$. By convention, every interpretation is a model for $\emptyset$[8]. It follows that every interpretation must be a model for $(Q \leftarrow ((P \vee Q) \wedge \neg P))$. Recall that this is just another way of stating that $(Q \leftarrow ((P \vee Q) \wedge \neg P))$ is valid (page 19)[9].

What is the difference between the concepts of logical consequence denoted by $\models$ and the connective $\rightarrow$ in a statement such as $\Sigma \models \Gamma$? where, $\Sigma = \{(P \wedge Q), (P \rightarrow R)\}$ and $\Gamma = \{P, Q, R\}$? And how do these two notions of implication relate to the phrase 'if....then', often used in propositions or theorems? We delineate the differences below:

[8] That is, we take the empty set to denote a distinguished proposition $True$ that is *true* in every interpretation. Correctly then, the formula considered is not $((\beta_1 \wedge \beta_2) \ldots \beta_n)))$ but $(True \wedge ((\beta_1 \wedge \beta_2) \ldots \beta_n))))$.

[9] To translate declarative knowledge into action (as in the case of the dog from Chrysippus's anecdote), one of two possible strategies can be adopted. The first is called 'Negative selection' which involves *excluding any provably futile* actions. The second is called 'Positive selection' which involves *suggesting only actions that are provably safe.* There can be some actions that are neither provably safe nor provably futile.

1. The connective $\rightarrow$ is a *syntactical symbol* called 'if ... then' or 'implication', which appears within formulæ. The truth value of the formula $(\alpha \rightarrow \xi)$ depends on the *particular* interpretation I we happen to be considering: according to the truth table, $(\alpha \rightarrow \xi)$ is true under I if α is false under I and/or ξ is true under I; $(\alpha \rightarrow \xi)$ is false otherwise.

2. The concept of 'logical consequence' or '(logical) implication', denoted by '$\models$' describes a *semantical relation* between formulæ. It is defined in terms of *all* interpretations: '$\alpha \models \xi$' is true if every interpretation that is a model of α, is also a model of ξ.

3. The phrase 'if. .. then', which is used when stating, for example, propositions or theorems is also sometimes called 'implication'. This describes a relation between assertions which are phrased in (more or less) natural language. It is used for instance in proofs of theorems, when we state that some assertion implies another assertion. Sometimes we use the symbols ' ' or . ¡=' for this. If assertion A implies assertion B, we say that B is a necessary condition for A (i.e., if A is true, B must necessarily be true), and A is a. Ilufficient condition for B (i.e., the truth of B is sufficient to make A true). In ('tum A implies B, and B implies A, we write "A iff B", where 'iff' abbreviates 'if, and only if'.

Closely related to logical consequence is the notion of *logical equivalence*. A pair of wffs α and β are logically equivalent means:

$$\alpha \models \beta \quad and \quad \beta \models \alpha$$

This means the truth values for α and β are the same in all cases, and is usually written more concisely as:

$$\alpha \equiv \beta$$

Examples of logically equivalent formulæ are provided by De Morgan's laws:

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

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

Also, if $True$ denotes the formula that is *true* in every interpretation and $False$ the formula that is *false* in every interpretation, then the following equivalences should be self-evident:

$$\alpha \equiv (\alpha \wedge True)$$

$$\alpha \equiv (\alpha \vee False)$$

More on the Conditional

We are mostly concerned with rules that utilise the logical connective $\leftarrow$. This makes this particular connective more interesting than the others, and it is worth noting some further details about it. Although we will present these here using examples from the propositional logic, the main points are just as applicable to formulæ in the predicate logic.

Recall the truth table for the conditional from page 18:

α	β	$(\alpha \leftarrow \beta)$
false	false	true
false	true	false
true	false	true
true	true	true

There is, therefore, only one interpretation that makes $(\alpha \leftarrow \beta)$ *false*. This may come as a surprise. Consider for example the statement:

$$\text{The earth is flat} \leftarrow \text{Humans are monkeys}$$

An interpretation that assigns *false* to both 'The earth is flat' and 'Humans are monkeys' makes this statement *true* (line 1 of the truth table). In fact, the only world in which the statement is false is one in which the earth is not flat, and humans are monkeys[10]. Consider now the truth table for $(\alpha \vee \neg\beta)$:

α	β	$\neg\beta$	$(\alpha \vee \neg\beta)$
false	false	true	true
false	true	false	false
true	false	true	true
true	true	false	true

It is evident from these truth tables that every model for $(\alpha \leftarrow \beta)$ is a model for $(\alpha \vee \neg\beta)$ and vice-versa. Thus:

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

Thus, the conditional:

$$(\text{Fred is human} \leftarrow (\text{Fred walks upright} \wedge \text{Fred has a large brain}))$$

is equivalent to:

[10]The unusual nature of the conditional is due to the fact that it allows premises and conclusions to be completely unrelated. This is not what we would expect from conditional statements in normal day-to-day discourse. For this reason, the $\leftarrow$ connective is sometimes referred to as the *material conditional* to distinguish it from a more intuitive notion.

$$(\text{Fred is human} \vee \neg\ (\text{Fred walks upright} \wedge \text{Fred has a large brain}))$$

Or, using De Morgan's Law (page 23) and dropping some brackets for clarity:

$$\text{Fred is human} \vee \neg\ \text{Fred walks upright} \vee \neg\ \text{Fred has a large brain}$$

In this form, each of the premises on the right-hand side of the the original conditional (Fred walks upright, Fred has a large brain) appear negated in the final disjunction; and the conclusion (Fred is human) is unchanged. For a reason that will become apparent later we will use the term *clauses* to denote formulæ that contain propositions or negated propositions joined together by disjunction ($\vee$). We will also use the term *literals* to denote propositions or negated propositions. Clauses are thus disjunctions of literals.

It is common practice to represent a clause as a set of literals, with the disjunctions understood. Thus, the clause above can be written as:

$$\{\ \text{Fred is human}, \neg\ \text{Fred walks upright}, \neg\ \text{Fred has a large brain}\ \}$$

The equivalence $\alpha \leftarrow \beta \equiv \alpha \vee \neg\beta$ also provides an alternative way of presenting the deduction theorem.

On page 22 the statement of this theorem was:

$$\Sigma \models \alpha \ \text{ if and only if } \ \Sigma - \{\beta_i\} \models (\alpha \leftarrow \beta_i)$$

This can now be restated as:

$$\Sigma \models \alpha \ \text{ if and only if } \ \Sigma - \{\beta_i\} \models (\alpha \vee \neg\beta_i)$$

The theorem thus operates as follows: when a formula moves from the left of $\models$ to the right, it is negated and disjoined (using $\vee$) with whatever exists on the right. The theorem can also be used in the "other direction": when a formula moves from the right of $\models$ to the left, it is negated and conjoined (using $\wedge$ or $\cup$ in the set notation) to whatever exists on the left. Thus:

$$\Sigma \models (\alpha \vee \neg\beta) \ \text{ if and only if } \ \Sigma \cup \{\neg\alpha\} \models \neg\beta$$

A special case of this arises from the use of the equivalence $\alpha \equiv (\alpha \vee \mathit{False})$ (page 23):

$$\Sigma \models \alpha \ \text{ if and only if } \ \Sigma \models (\alpha \vee \mathit{False}) \ \text{ if and only if } \ \Sigma \cup \{\neg\alpha\} \models \mathit{False}$$

The formula False is commonly written as $\Box$ and the result above as:

$$\Sigma \models \alpha \ \text{ if and only if } \ \Sigma \cup \{\neg\alpha\} \models \Box$$

The conditional $(\alpha \leftarrow \beta)$ is sometimes mistaken to mean the same as $(\alpha \wedge \beta)$. Comparison against the truth table for $(\alpha \wedge \beta)$ shows that these two formulæ are not equivalent:

α	β	$(\alpha \wedge \beta)$
false	false	false
false	true	false
true	false	false
true	true	true

There are a number of ways in which $(\alpha \leftarrow \beta)$ can be translated in English. Some of the more popular ones are:

If β, then α	α, if β	β implies α
β only if α	β is sufficient for α	α is necessary for β
All β's are α's		

Note the following related statements:

Conditional	$(\alpha \leftarrow \beta)$
Contrapositive	$(\neg\beta \leftarrow \neg\alpha)$

It should be easy to verify the following equivalence:

$$\text{Conditional} \equiv \text{Contrapositive} \qquad (\alpha \leftarrow \beta) \equiv (\neg\beta \leftarrow \neg\alpha)$$

Errors of reasoning arise by assuming other equivalences. Consider for example the pair of statements:

$$S_1 : \text{Fred is an ape} \leftarrow \text{Fred is human}$$
$$S_2 : \text{Fred is human} \leftarrow \text{Fred is an ape}$$

S_2 is the sometimes called the *converse* of S_1. An interpretation that assigns *true* to 'Fred is an ape' and *false* to 'Fred is human' is a model for S_1 but not a model for S_2. The two statements are thus not equivalent.

More on Normal Forms

We are now able to state two properties concerning normal forms:

1. If F is a formula in CNF and G is a formula in DNF, then $\neg F$ is a formula in DNF and $\neg G$ is a formula in CNF. This is a generalisation of De Morgan's laws and can be proved using the technique of mathematical induction (that is, show truth for a formula with a single literal; assume truth for a formula with n literals; and then show that it holds for a formula with $n + 1$ literals.)

2. Every formula F can be written as a formula F_1 in CNF and a formula F_2 in DNF. It is straightforward to see that any formula F can be written as a DNF formula by examining the rows of the truth table for F for which F is true. Suppose F consists of the propositions $A_1, A_2, \ldots, A_n$. Then each such row is equivalent to some conjunction of literals $L_1, L_2, \ldots, L_n$, where L_i is equal to A_i if A_i is true in that row and equal to $\neg A_i$ otherwise. Clearly, the disjunction of each row for which F is true gives the DNF formula for F. We can get the corresponding CNF formula G by negating the DNF formula (using the property above), or by examining the rows for which F is false in the truth table.

It should now be clear that a CNF expression is nothing more than a conjunction of a set of clauses (recall a clause is simply a disjunction of literals). It is therefore possible to convert any propositional formula F into CNF—either using the truth table as described, or using the following procedure:

1. Replace all conditionals statements of the form $A \leftarrow B$ by the equivalent form using disjunction (that is, $A \vee \neg B$). Similarly replace all $A \leftrightarrow B$ with $(A \vee \neg B) \wedge (\neg A \vee B)$.

2. Eliminate double negations ($\neg\neg A$ replaced by A) and use De Morgan's laws wherever possible (that is, $\neg(A \wedge B)$ replaced by $(\neg A \vee \neg B)$ and $\neg(A \vee B)$ replaced by $(\neg A \wedge \neg B)$).

3. Distribute the disjunct $\vee$. For example, $(A \vee (B \wedge C))$ is replaced by $(A \vee B) \wedge (A \vee C)$.

An analogous process converts any formula to an equivalent formula in DNF. We should note that during conversion, formulæ can expand exponentially. However, if only satisfiability should be preserved, conversion to CNF formula can be done polynomially.

1.3.3 Inference

Enumerating and comparing models is one way of determining whether one formula is a logical consequence of another. While the procedure is straightforward, it can be tedious, often requiring the construction of entire truth tables. A different approach makes no explicit reference to truth values at all. Instead, if α is a logical consequence of Σ, then we try to show that we can *infer* or *derive* α from Σ using a set of well-understood rules. Step-by-step application of these rules results in a *proof* that deduces that α follows from Σ. The rules, called *rules of inference*, thus form a system of performing calculations with propositions, called the *propositional calculus*[11]. Logical implication can be mechanized by using a propositional calculus. We will first concentrate on a particular inference rule called *resolution*.

[11] In general, a set of inference rules (potentially including so called logical axioms) is called a *calculus*.

1.3.4 Resolution

Before proceding further, some basic terminology from proof theory may be helpful (this is not specially confined to the propositional calculus). Proof theory considers the *derivability* of formulæ, given a set of inference rules $\mathcal{R}$. Formulæ given initially are called *axioms* and those derived are *theorems*. That formula α is a theorem of a set of axioms Σ using inference rules $\mathcal{R}$ is denoted by:

$$\Sigma \vdash_{\mathcal{R}} \alpha$$

When $\mathcal{R}$ is obvious, this is simply written as $\Sigma \vdash \alpha$. The axioms can be valid (that is, all interpretations are models), or problem-specific (that is, only some interpretations may be models). The axioms together with the inference rules constitute what is called an *inference system*. The axioms together with all the theorems that are derivable from it is called a *theory*. A theory is said to be *inconsistent* if there is a formula α such that the theory contains both α and $\neg\alpha$.

We would like the theorems derived to be logical consequences of the axioms provided. For, if this were the case then by definition, the theorems will be *true* in all models of the axioms (recall that this is what logical consequence means). They will certainly be true, therefore, in any particular 'intended' interpretation of the axioms. Ensuring this property of the theorems depends entirely on the inference rules chosen: those that have this property are called *sound*. That is:

$$\text{if } \Sigma \vdash_{\mathcal{R}} \alpha \text{ then } \Sigma \models \alpha$$

Some well-known sound inference rules are:

$$\begin{array}{ll} \text{Modus ponens:} & \{\beta, \alpha \leftarrow \beta\} \vdash \alpha \\ \text{Modus tollens:} & \{\neg\alpha, \alpha \leftarrow \beta\} \vdash \neg\beta \end{array}$$

Theorems derived by the use of sound inference rules can be added to the original set of axioms. That is, given a set of axioms Σ and a theorem α derived using a sound inference rule, $\Sigma \equiv \Sigma \cup \{\alpha\}$.

We would also like to derive *all* logical consequences of a set of axioms and rules with this property at said to be *complete*:

$$\text{if } \Sigma \models \alpha \text{ then } \Sigma \vdash_{\mathcal{R}} \alpha$$

Axioms and inference rules are not enough: we also need a strategy to select and apply the rules. An inference system (that is, axioms and inference rules) along with a strategy is called a *proof procedure*. We are especially interested here in a special inference rule called *resolution* and a strategy called SLD (the meaning of this is not important at this point: we will come to it later). The result is a proof procedure called *SLD-resolution*. Here we will simply illustrate the rule of resolution for manipulating propositional formulæ, and use an unconstrained proof strategy. A description of SLD will be left for a later section.

Suppose we are given as axioms the conditional formulæ (using the informal notation that replaces $\wedge$ with commas):

β_1 : Fred is an ape $\leftarrow$ Fred is human

β_2 : Fred is human $\leftarrow$ Fred walks upright, Fred has a large brain

Then the following is a theorem resulting from the use of resolution:

α : Fred is an ape $\leftarrow$ Fred walks upright, Fred has a large brain

That α is indeed a logical consequence of $\beta_1 \wedge \beta_2$ can be checked by constructing truth tables for the formulæ: you will find that every interpretation that makes $\beta_1 \wedge \beta_2$ *true* will also make α *true*. More generally, here is the rule of resolution when applied to a pair of conditional statements:

$$\{(P \leftarrow Q_1, \ldots, Q_i, \ldots Q_n),\ (Q_i \leftarrow R_1, \ldots R_m)\} \quad \vdash$$
$$P \leftarrow Q_1, \ldots, Q_{i-1}, R_1, \ldots R_m, Q_{i+1}, \ldots, Q_n$$

The equivalence from page 24 ($\alpha \leftarrow \beta \equiv \alpha \vee \neg\beta$) allows resolution to be presented in a different manner (we have taken the liberty of dropping some brackets here):

$$\{(P \vee \neg Q_1 \vee \ldots \neg Q_i \vee \ldots \vee \neg Q_n),\ (Q_i \vee \neg R_1 \vee \ldots \vee \neg R_m)\} \quad \vdash$$
$$P \vee \neg Q_1 \vee \ldots \vee \neg Q_{i-1} \vee \neg R_1 \vee \ldots \vee \neg R_m \vee \neg Q_{i+1} \vee \ldots \vee \neg Q_n$$

On page 25, we introduced the terms clauses and literals. Thus, resolution applies to a pair of 'parent' clauses that contain *complementary* literals $\neg L$ and L. The result (the 'resolvent') is a clause containing all literals from each clause, except the complementary pair. Or, more abstractly, let C_1 and C_2 be a pair of clauses, and let $L \in C_1$ and $\neg L \in C_2$. Then, the resolvent of C_1 and C_2 is the clause:

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

Resolution of a pair of *unit clauses*—those that contain just single literals L and $\neg L$—results in the the *empty clause*[12], or $\Box$, which means that the parent clauses were inconsistent.

We can show that resolution is a sound inference rule.

Theorem 7 *Suppose R is the resolvent of clauses C_1 and C_2. That is, $\{C_1, C_2\} \vdash R$. The resolution is sound, that is, $\{C_1, C_2\} \models R$.*

Proof:: We want to show that if C_1 and C_2 are true and R is a resolvent of C_1 and C_2 then R is true. Let us assume C_1 and C_2 are true, and that R was obtained by resolving on some literal L in C_1 and C_2. Further, let $C_1 = C \vee L$

[12]Note the difference between an empty clause $\Box$ and empty set of clauses $\{\}$. An interpretation I logically entails C *iff* there exists an $l \in C$ such that $I \models l$. I logically entails Σ if for all $C \in \Sigma$, $I \models C$. Thus, by definition, for all interpretations I, $I \not\models \Box$ and $I \not\models \{\Box\}$, whereas $I \models \{\}$.

and $C_2 = D \vee \neg L$, giving $R = C \vee D$. Now, L is either true or false. Suppose L is true. Then clearly C_1 is true, but since $\neg L$ is false and C_2 is true (by assumption), it must be that D, and hence R must be true. It is easy to see that we similarly arrive to the same conclusion about R even if L was false. $\Box$

So, the theorems obtained by applying resolution to a set of axioms are all logical consequences of the axioms. In general, we will denote a clause C derived from a set of clauses Σ using resolution by $\Sigma \vdash_R C$. This means that there is a finite sequences of clauses $R_1, \ldots, R_k = C$ such that each C_i (where C_i is a clause being resolved upon in the i^{th} resolution step) is either in Σ or is a resolvent of a pair of clauses already derived (that is, from $\{R_1, \ldots, R_{i-1}\}$) . Now, although it is the case that if $\Sigma \vdash_R \alpha$ then $\Sigma \models \alpha$, the reverse does not hold. For example, a moment's thought should convince you that:

$$\{\text{Fred is an ape, Fred is human }\} \models \text{Fred is an ape} \leftarrow \text{Fred is human}$$

However, using resolution, there is no way of deriving Fred is an ape $\leftarrow$ Fred is human from Fred is an ape and Fred is human. As an inference rule, resolution is thus incomplete[13]. However, it does have an extremely useful property known as *refutation completeness*. This is that if a formula Σ is inconsistent, then the empty clause $\Box$ will be eventually derivable by resolution. We will distinguish between the two completeness by referring to general completeness as affirmation completeness.

Thus, since Fred is an ape $\leftarrow$ Fred is human is a logical consequence of Fred is an ape and Fred is human, then the formula:

$$\Sigma : \{\text{Fred is an ape, Fred is human, } \neg(\text{Fred is an ape} \leftarrow \text{Fred is human})\}$$

must be inconsistent. This can be verified using resolution. First, the clausal form of (Fred is an ape $\leftarrow$ Fred is human) is (Fred is an ape $\vee \neg$ Fred is human). Using De Morgan's Law on this clausal form, we can see that $\neg$(Fred is an ape $\leftarrow$ Fred is human) is equivalent to $\neg$ Fred is an ape $\wedge$ Fred is human. We can now rewrite Σ :

$$\Sigma' : \{\text{Fred is an ape, Fred is human, } \neg\text{Fred is an ape}\}$$

Resolution of the pair Fred is an ape, $\neg$Fred is an ape would immediately result in the empty clause $\Box$. The general steps in a refutation proof procedure using resolution are therefore:

- Let S be a set of clauses and α be a propositional formula. Let $C = S \cup \{\neg\alpha\}$.
- Repeatedly do the following:
    1. Select a pair of clauses C_1 and C_2 from C that can be resolved on some proposition P.

[13] It can be however proved that resolution is affirmation complete with respect to atomic conclusions.

2. Resolve C_1 and C_2 to give R.
3. If $R = \Box$ then stop. Otherwise, if R contains both a proposition Q and its negation $\neg Q$ then discard R. Otherwise add R to C.

In general, we know that any formula F can be converted to a conjunction of clauses. We can distinguish between the following sets. $Res^0(F)$, which is simply the set of clauses in F. $Res^n(F)$, for $n > 0$, which is the clauses containing all clauses in $Res^{n-1}(F)$ and all clauses obtained by resolving a pair of clauses from $Res^{n-1}(F)$. Since there are only a finite number of propositional symbols in F and a finite number of clauses in its CNF, we can see that there will only be a finite number of clauses that can be obtained using resolution. That is, there is some m such that $Res^m(F) = Res^{m-1}(F)$. Let us call this final set consisting of all the original clauses and all possible resolvents $Res^*(F)$. Then, the property of refutation-completeness for resolution can be stated more formally as follows:

Theorem 8 *For some formula F, $\Box \in Res^*(F)$ if and only if F is unsatisfiable.*

Though the resolution rule by itself is not (affirmation) complete for clauses in general, this property states that it is complete with respect to unsatisfiable sets of clauses. The complete proof of this will be provided on page 33. To get you started however, we show that if $\Box \in Res^*(F)$ then F is unsatisfiable. We can assume that $\Box \notin Res^0(F)$, since $\Box$ is not a disjunction of literals. Therefore there must be some k for which $\Box \notin Res^k(F)$ and $\Box \in Res^{k+1}(F)$. This can only mean that both L and $\neg L$ are in $Res^k(F)$. That is L and $\neg L$ are obtained from F by resolution. By the property of soundness of resolution, this means that $F \models (L \wedge \neg L)$. That is, F is unsatisfiable.

There are also other proof processes that are refutation-complete. Examples of such processes are the Davis-Putnam Procedure[14], Tableaux Procedure, *etc.* In the worst case, the resolution search procedure can take exponential time. This, however, very probably holds for all other proof procedures. For CNF formulae in propositional logic, a type of resolution process called the Davis-Putnam Procedure (backtracking over all truth values) is probably (in practice) the fastest refutation-complete process.

The Subsumption Theorem

A property related to logical implication is that of subsumption. A propositional clause C *subsumes* a propositional clause D if $C \subseteq D$. What does this mean? It just means that every literal in C appears in D. Here are a pair of clauses C and D such that C subsumes D:

$$C : \mathsf{Fred\ is\ an\ ape}$$

$$D : \mathsf{Fred\ is\ an\ ape} \leftarrow \mathsf{Fred\ is\ human}$$

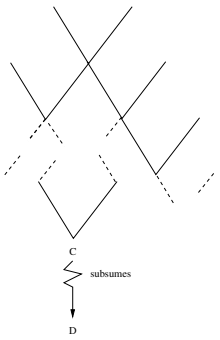
[14]It can be proved that the Davis-Putnam procedure is sound as well as complete.

In general, it should be easy to see that if C and D are clauses such that $C \subseteq D$, then $C \models D$. In fact, for propositional logic, it is also the case that if $C \models D$ then C subsumes D (we see why this is so shortly).

The notion of subsumption acts as the basis for an important result linking resolution and logical implication, called the *subsumption theorem*:

Theorem 9 *If Σ is a set of clauses and D is a clause. Then $\Sigma \models D$ if and only if D is a tautology or there is a clause C such that there is a derivation of C from Σ using resolution ($\Sigma \vdash_R C$) and C subsumes D.*

By "derivation of a clause C" here, we mean the same as on page 30, that is, there is a sequence of clauses $R_1, \ldots, R_k = C$ such that each R_i is either in Σ or is a resolvent of a pair of clauses in $\{R_1, \ldots, R_{i-1}\}$. In effect, the Subsumption Theorem tells us that logical implication can be decomposed into a sequence of resolution steps, followed by a subsumption step:



Proof of Subsumption Theorem:

We will show that "only if" part of the theorem holds using the method of induction on the size of Σ:

1. Let $|\Sigma| = 1$. That is $\Sigma = \{\alpha_1\}$. Let $\Sigma \models D$. Since the only result of applying resolution to Σ is α_1, we need to show that $\alpha_1 \subseteq D$. Suppose $\alpha_1 \not\subseteq D$. Let L be a literal in α_1 that is not in D. Let I be an interpretation that assigns L to true and all literals in D to false. Clearly I is a model for α_1 but not a model for D, which is not possible since $\Sigma \models D$. Therefore, $\alpha_1 \subseteq D$.

2. Let the theorem hold for $|\Sigma| = n$. We will see that it follows that it holds for $|\Sigma| = n + 1$. Let $\Sigma = \{\alpha_1, \ldots, \alpha_{n+1}\}$ and $\Sigma \models D$. By the Deduction Theorem, we know that this means $\Sigma - \{\alpha_{n+1}\} \models (D \leftarrow \alpha_{n+1})$, or $\Sigma - \{\alpha_{n+1}\} \models (D \vee \neg\alpha_{n+1})$. Let us set $\Sigma' = \Sigma - \{\alpha_{n+1}\}$, and let $L_1, \ldots, L_k$ be the literals in α_{n+1} that do not appear in D. That is $\alpha_{n+1} = L_1 \vee \cdots \vee L_k \vee D'$, where $D' \subseteq D$. $\Sigma' \models (D \vee \neg\alpha_{n+1})$, you should be able to see that $\Sigma' \models (D \vee \neg L_i)$ for $1 \leq i \leq k$. Since $|\Sigma'| = n$ and we

believe, by the induction hypothesis, that the subsumption theorem holds for $|\Sigma'| = n$, there must be some β_i for each L_i, such that $\Sigma' \vdash_R \beta_i$ and $\beta_i \subseteq (D \vee \neg L_i)$. Suppose $\neg L_i \notin \beta_i$. Then $\beta_i \subseteq D$, which means that β_i subsumes D. Since $\Sigma' \models \beta_i$ and $\Sigma \models \Sigma'$, the result follows. Now suppose $\neg L_i \in \beta_i$. That is, $\beta_i = \neg L_i \vee \beta_i'$, where $\beta_i' \subseteq D$. Clearly, we can resolve this with $\alpha_{n+1} = L_1 \vee \cdots \vee L_i \vee \cdots \vee L_k \vee D'$ to give $L_1 \vee \cdots \vee L_{i-1} \vee \beta_i' \vee \cdots \vee L_k \vee D'$. Progressively resolving against each of the β_i, we will be left with the clause $C = \beta_1' \vee \beta_2' \vee \cdots \vee \beta_k' \vee D'$. Since C is the result of resolutions using a clause from Σ (that is α_{n+1}) and clauses derivable from $\Sigma' \subset \Sigma$, it is evident that $\Sigma \vdash_R C$. Also, since $\beta_i' \subseteq D$ and $D' \subseteq D$, $C \subseteq D$ and the result follows.

You should be able to see that the proof in the other direction (the "if" part) follows easily enough from the soundness of resolution. □

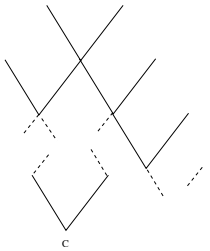
An immediate consequence of the Subsumption Theorem is that refutation-completeness of resolution follows.

Proof of Theorem 8

Recall what refutation completeness of resolution means: if Σ is a set of clauses that is unsatisfiable, then the empty clause □ is derivable using resolution. If Σ is unsatisfiable, then $\Sigma \models \square$. From the Subsumption Theorem, we know that if $\Sigma \models \square$, there must be a clause C such that $\Sigma \vdash_R C$ and C subsumes □. But the only clause subsuming □ is □ itself. Hence $C = \square$, which means that if $\Sigma \models \square$ then $\Sigma \vdash_R \square$.

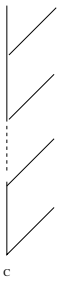
Proofs Using Resolution

So far, we have no strategy for directing the clauses obtained using resolution. Clauses are derived using any pair of clauses with complementary literals, and the process simply continues until we find the clause we want (for example, □ if we are interested in a proof of unsatisfiability). This procedure is clearly quite inefficient, since there is almost nothing constraining a proof, other than the presence of complementary literals. A "proof" for a clause C then ends up looking something like this:



Being creatures of limited patience and resources, we would like more directed approach. We can formalise this by changing our notion of a derivation. Recall

what we have been using so far: the derivation of a clause C from a set of clauses Σ means there is a sequence of clauses $R_1, \ldots, R_k = C$ such that each C_i is either in Σ or is a resolvent of a pair of clauses in $\{R_1, \ldots, R_{i-1}\}$. This results in the unconstrained form of a proof for C. We will say that there is a *linear* derivation for C from Σ if there is a sequence $R_0, \ldots, R_k = C$ such that $R_0 \in \Sigma$ and each R_i $(1 \leq i \leq k)$ is a resolvent of R_{i-1} and a clause $C_i \in \Sigma \cup \{R_0, \ldots, R_{i-2}\}$. With a little thought, you should be able to convince yourself that this will result in a derivation with a "linear" look:



Here, you can see that each new resolvent forms one of the clauses for the next resolution step. The other clause—sometimes called the "side clause"—can be any one of the clauses in Σ or a previous resolvent. For reasons evident from the diagram above, the proof strategy is called linear resolution, and we will extend our notation to indicate both the inference rule and the proof strategy. Thus $\Sigma \vdash_{LR} C$ will mean that C is derived from Σ using linear resolution. We can restrict things even further, by requiring side clauses to be only from Σ. The resulting proof strategy, called *input resolution* is important as it is a generalised form of SLD resolution, first mentioned on page 28.

While the restrictions imposed by the proof strategies ensure that proofs are more directed (and hence efficient), it is important at this point to ask: at what cost? Of course, since we are still using resolution as an inference rule, the individual (and overall) inference steps remain sound. But what about completeness? By this we mean refutation-completeness, since this is the only kind of completeness we were able to show with unconstrained resolution. In fact, it is the case that linear resolution retains the property of refutation-completeness, but input resolution for arbitrary clauses does not. That input resolution is not refutation complete can be proved using a simpe counter-example:

$$C_0 : \mathsf{Fred\ is\ an\ ape} \leftarrow \mathsf{Fred\ is\ human}$$
$$C_1 : \mathsf{Fred\ is\ an\ ape} \leftarrow \neg\ \mathsf{Fred\ is\ human}$$
$$C_2 : \neg\ \mathsf{Fred\ is\ an\ ape} \leftarrow \mathsf{Fred\ is\ human}$$
$$C_3 : \neg\ \mathsf{Fred\ is\ an\ ape} \leftarrow \neg\ \mathsf{Fred\ is\ human}$$

Now, a little effort should convince you that this set of clauses is unsatisfiable. But input resolution will simply yield a sequence of resolvents: Fred is human, Fred is an ape, Fred is human, ...

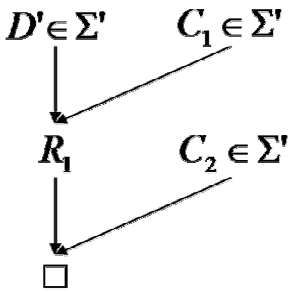


Figure 1.4: Part of case 1 of the proof for theorem 10.

Theorem 10 *If Σ is an unsatisfiable set of clauses, and $C \in \Sigma$ such that $\Sigma \backslash \{C\}$ is satisfiable, then there is a linear refutation of Σ with C as top clause.*

Proof:

We can assume Σ is finite. Let n be the number of distinct atoms occurring in literals in literals in Σ. We prove the lemma by induction on n

1. If $n = 0$, then $\Sigma = \{\Box\}$. Since $\Sigma \setminus \{C\}$ is satisfiable, $C = \Box$.

2. Suppose the lemma holds for $n \leq m$, and suppose $m + 1$ distinct atoms appear in Σ. We distinguish two cases.

   - *Case 1*: Suppose $C = L$, where L is a literal. We first delete all clauses from Σ which contain the literal L (so we also delete C itself from Σ). Then we replace clauses which contain the literal $\neg L$ by clauses constructed by deleting these $\neg L$ (so for example, $L_l \vee \neg L \vee L_2$ will be replaced by $L_1 \vee L_2$). Call the finite set obtained in this way Γ.

     Note that neither the literal L, nor its negation, appears in clauses in Γ. If M were a Herbrand model of Γ, then $M \cup \{L\}$ (*i.e.*, the Herbrand interpretation which makes L true, and is the same as M for other literals) would be a Herbrand model of Σ. Thus since Σ is unsatisfiable, Γ must be unsatisfiable.

     Now let Σ' be an unsatisfiable subset of Γ, such that every proper subset of Σ' is satisfiable. Σ' must contain a clause D' obtained from a member of Σ which contained $\neg L$, for otherwise the unsatisfiable set Σ' would be a subset of $\Sigma \backslash \{C\}$, contradicting the assumption that $\Sigma \setminus \{C\}$ is satisfiable. By construction of Σ', we have that $\Sigma' \setminus \{D'\}$ is satisfiable. Furthermore, Σ' contains at most m distinct atoms, so by the induction hypothesis there exists a linear refutation of Σ' with top clause D'. See the Figure 1.4 for illustration.

     Each side clause in this refutation that is not equal to a previous center clause, is either a member of Σ or is obtained from a member

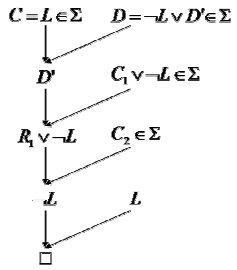


Figure 1.5: The complete picture of case 1 of the proof for theorem 10.

of Σ by means of the deletion of $\neg L$. In the latter kind of side clauses, put back the deleted $\neg L$ literals, and add these $\neg L$ to all later center clauses. Note that afterwards, these center clauses may contain multiple copies of $\neg L$. In particular, the last center clause changes from $\Box$ to $\neg L \vee \ldots \vee \neg L$. Since D' is a resolvent of C and $D = \neg L \vee D' \in \Sigma$, we can add C and D as parent clauses on top of the previous top clause D'. That way, we get a linear derivation of $\neg L \vee \ldots \vee \neg L$ from Σ, with top clause C. Finally, the literals in $\neg L \vee \ldots \vee \neg L$ can be resolved away using the top clause $C = L$ as side clause. This yields a linear refutation of Σ with top clause C (see Figure 1.5).

- *Case 2:* Exercise

□

The incompleteness of input also means that the Subsumption Theorem will not hold for input resolution in general. What, then, can we say about SLD resolution? The short answer is that it too is incomplete. But, for a restricted form of clauses, input and SLD resolution *are* complete. The restriction is to Horn clauses: recall that these are clauses that have at most 1 positive literal. Indeed, it is this restriction that forms the basis of theorem-proving in the PROLOG language, which is restricted (at least in its pure form) to Horn clauses, albeit in first-order logic (but the result still holds in that case as well).

Proofs Using SLD Resolution

Before we get to SLD, we first make our description of input resolution a little more precise: the derivation of a clause C from a set of clauses Σ using input resolution means there is a sequence of clauses $R_0, \ldots, R_k = C$ such that $R_0 \in \Sigma$ and each R_i $(1 \leq i \leq k)$ is a resolvent of R_{i-1} and a clause $C_i \in \Sigma$. Now, we add further restrictions. Let Σ be a set of Horn clauses. Further, let R_i be a resolvent of a selected negative literal in R_{i-1} and the positive literal of a definite clause $C_i \in \Sigma$. The selection rule is called the "computation rule" and

the resulting proof strategy is called SLD resolution ("Selected Linear Definite" resolution). We illustrate this with an example. Let Σ be the set of clauses:

C_0 : $\neg$ Fred is an ape

C_1 : Fred is an ape $\leftarrow$ Fred is human, Fred has hair

C_2 : Fred is human

C_3 : Fred has hair

A little thought should convince you that $\Sigma \models \Box$. We want to see if $\Sigma \vdash_{SLD} \Box$. It is evident that C_0 and C_1 resolve. The resolvent is R_1:

C_0 : $\neg$ Fred is an ape

C_1 : Fred is an ape $\leftarrow$ Fred is human, Fred has hair

R_1 : $\leftarrow$ Fred is human, Fred has hair

C_2 : Fred is human

C_3 : Fred has hair

Since we are using SLD, one of the resolvents for the next step has to be R_1. The other resolvent has to be one of the C_i's. Suppose our selection rule selects the "rightmost" literal first for resolution (that is, $\neg$ Fred has hair in R_1) This resolves with C_3, giving R_2 : $\neg$ Fred is human, which in turn resolves with C_2 to give $\Box$. The SLD (input) resolution diagram for this is presented in Figure 1.6.

It is more common, especially in the logic-programming literature, to present instead the search process confronting a SLD-resolution theorem prover in the form of a tree-diagram, called an *SLD-tree*. Such a tree effectively contains all possible derivations that can be obtained using a particular literal selection rule. Each node in the tree is a "goal" of the form $\leftarrow L_1, L_2, \ldots, L_k$. That is, it is a clause of the form $(\neg L_1 \vee \neg L_2 \vee \cdots \vee \neg L_k)$. Given a set of clauses Σ, the children of a node in the SLD-tree are the result of resolving with clauses in Σ (nodes representing the empty clause $\Box$ have no children). The SLD-tree for the example we just looked at is shown in Figure 1.7

We can now see what refutation-completeness for Horn clauses for SLD-resolution means in terms of SLD-trees. In effect, this means that if a set of clauses is unsatisfiable then there will be a leaf in the SLD-tree with the empty clause $\Box$. Further, the computation rule will not alter this (informally, you can see that different computation rules will simply move the location of the $\Box$ around). We will have more to say on SLD-resolution with first-order logic in a later section. There, we will see that in addition to the computation rule, we will also need a "search" rule that determines how the SLD-tree is searched. Search trees there can have infinite branches, and although completeness is unaffected by the choice of the computation rule (that is, there will be a $\Box$ in the tree if the set of first-order clauses is unsatisfiable), we may not be able to reach it with a fixed search rule.

Theorem 11 *If Σ is an unsatisfiable set of horn clauses, then there is an SLD refutation of Σ.*

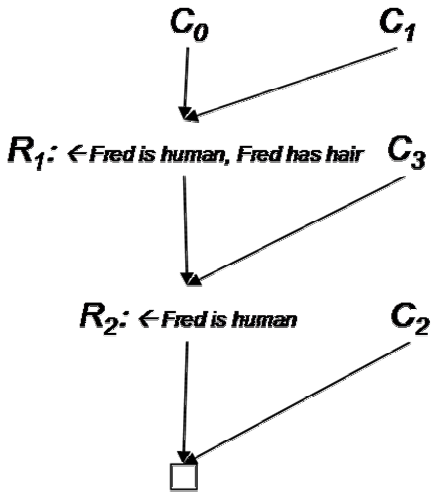


Figure 1.6: Example of SLD-deduction of $\square$ from Σ.

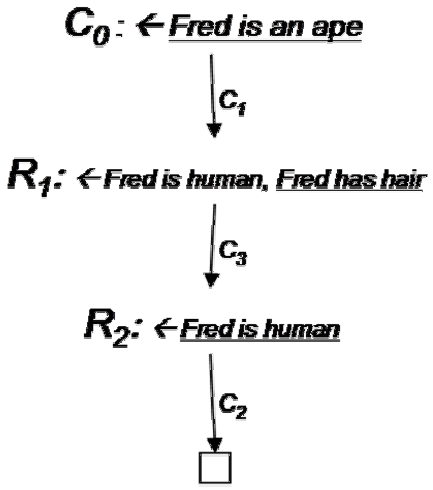


Figure 1.7: Example SLD-tree with C_0 at root.

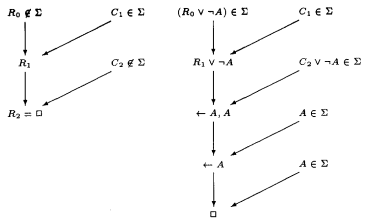


Figure 1.8: Illustration of the proof for theorem 11. The SLD refutation of Σ' is on the left and that for Σ is on the right.

Proof:

We can assume Σ is finite. Let n be the number of facts (clauses consisting of a single positive literal) in Σ. We prove the lemma by induction on n

1. If $n = 0$, then $\square \in \Sigma$ for otherwise the empty set would be a Herbrand model of I.

2. Suppose the lemma holds for $0 \leq n \leq m$, and suppose $m+1$ distinct facts appear in Σ. If $\square \in \Sigma$ the lemma is obvious, so suppose $\square \notin \Sigma$.

   Let, A be a fact in Σ. We first delete all clauses from Σ; which have A as head (so we also delete the fact A from Σ). Then we replace clauses which have A in their body by clauses constructed by deleting these atoms A from the body (so for example, $B \leftarrow A, B_1, \ldots, B_k$ will be replaced by $B \leftarrow B_1, \ldots, B_k$). Call the set obtained in this way Σ'. If M is a model of Σ', then $M \cup \{A\}$ is a Herbrand model of Σ. Thus since Σ is unsatisfiable, Σ' must be unsatisfiable. Σ' only contains m facts, so by the induction hypothesis, there is an SLD-refutation of Σ'. If this refutation only uses clauses from Σ' which were also in Σ then this is also an SLD-refutation of Σ, so then we are done. Otherwise, if C is the top clause or an input clause in this refutation and $C \notin \Sigma$, then C was obtained from some $C' \in \Sigma$ by deleting all atoms A from the body of Σ. For all such C, do the following: restore the previously deleted copies of A to the body of C (which turns C into C' again), and add these atoms A to all later resolvents. This way, we can turn the SLD-refutation of Σ' into an SLD-derivation of $\leftarrow A, \ldots, A$ from Σ. See Figure 1.8 for illustration, where we add previously deleted atoms A to the bodies of R_0 and C_2. Since also $A \in \Sigma$, we can construct an SLD-refutation of Σ, using A a number of times as input clause to resolve away all members of the goal $\leftarrow A, \ldots, A$.

□

1.3.5 Davis-Putnam Procedure

The inference problem addressed so far (particularly through the resolution procedure) is to determine if a proposition α logically follows from a given logical theory Σ. As we saw, this is achieved by reducing the problem to a coNP-complete[15] unsatisfiability problem; based on the contradiction theorem, it amounts to negating the goal formula α, add it to the theory and test the conjunction for unsatisfiability.

However, often, one is faced with the requirement for a model M for a logical theory Σ. This can turn out to be easier problem than the usual problem of inference, since it is enough to find one model for the theory, as against trying all possible truth assignments as in the case of solving an unsatisfiability problem. For example, theory might describe constraints on the different parts of a car. And you are interested in a model that satisifes all the constraints. In terms of search, you search the space of assignment and stop when you find an assignment that satisifies the theory.

While the resolution procedure can be modified so that it gives you a model, the Davis-Putnam-Logemann-Loveland (DPLL) procedure is a more efficient procedure for solving SAT problems. Given a set of clauses Σ defined over a set of variables $\mathcal{V}$, the Davis-Putnam procedure $DPLL(\Sigma)$ returns 'satisfiable' if is satisfiable. Otherwise return 'unsatisfiable'.

The $DPLL(\Sigma)$ procedure consists of the following steps. The first two steps specify termination conditions. The last two rules actually work on the clauses in Σ.

1. If $\Sigma = \emptyset$, return 'satisfiable'. This convention was introduced on pages 22 when we introduced logical entailment, as also in the footnote on page 29.

2. If $\square \in \Sigma$ return 'unsatisfiable'. This convention was discussed in the footnote on page 29.

3. **Unit-propagation Rule:** If Σ contains any unit-clause $C = \{c\}$ (*c.f.* page 29 for definition), assign a truth-value to the variable in literal c that satisfies c, 'simplify' Σ to Σ' and (recursively) return $DPLL(\Sigma')$. The rationale here is that if Σ has any unit clase $C = \{c\}$, the only way to satisfy C is to make c true. The simplification of Σ to Σ' is achieved by:

   (a) removing all clauses from Σ that contain the literal c (since all such clauses will now be true)

   (b) removing the negation of literal c (*i.e.*, $\neg c$) from every clause in Σ that contains it

[15] A decision problem C is Co-NP-complete if it is in Co-NP and if every problem in Co-NP is polynomial-time many-one reducible to it. The problem of determining whether a given boolean formula is tautology is a coNP-complete problem as well. A problem C is a member of co-NP if and only if its complement $\overline{C}$ is in complexity class NP. For example, the satisfiability problem is an NP-complete problem. Therefore the unsatisfiability problem is a coNP-complete problem.

4. **Splitting Rule:** Select from $\mathcal{V}$, a variable v which has not been assigned a truth-value. Assign one truth value t to it, simplify Σ to Σ' and (recursively) call $DPLL(\Sigma')$.

   (a) If the call returns 'satisfiable' (*i.e.*, we made a right choice for truth value of v), then return 'satisfiable'.

   (b) Otherwise (that is, if we made a wrong choice for truth value of v), assign the other truth-value to v in Σ, simplify to Σ'' and return $DPLL(\Sigma'')$.

The DPLL procedure can construct a model (if there exists one) by doing a book-keeping over all the assignments. This procedure is complete (that is, it constructs a model if there exists one), correct (the procedure always finds a trutn assignment that is a model) and guaranteed to terminate (since the space of possible assignments is finite and since DPLL explores that space systematically). In the worst case, DPLL requires exponential time, owing to the splitting rule. This is not surprissing, given the NP-completeness of the SAT problem. Heuristics are needed to determine (i) which variable should be instantiated next and (ii) to what value the instantiated varaible should be set. In all SAT competitions[16] so far, Davis Putnam-based procedures have shown the best performance.

As an illustration, we will use the DPLL procedure to determine a model for $\Sigma = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}\}$. Since Σ is neither an empty set nor contains the empty clause, we move on to step 3 of the procedure. Σ contains a single unit clause $\{c\}$, which we will set to true and simplify the theory to $\Sigma^1 = \{\{a, b\}, \{\neg a, \neg b\}, \{a, \neg b\}\}$. Next, we apply the splitting rule. Let us choose a and set it to 'false'. This yields $\Sigma^2 = \{\{b\}, \{\neg b\}\}$. It can be verified that $\Sigma^2 \equiv \{\Box\}$. We therefore backtrack and set a to 'true'. This yields $\Sigma^3 = \{\{\neg b\}\}$. Thereafter, application of unit propagation yields $\Sigma^4 = \{\}$, which is satisfiable according to step 1 of the procedure. Thus, using the DPLL procedure, we obtain a model for Σ as $M = \{a, c\}$.

As another example, consider $\Sigma = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$. As we will see, application to DPLL to this problem does not involve any backtracking, making the task relatively easier. Σ is neither an empty set nor contains the empty clause. Hence we apply the unit propagation rule 3 on $\{d\}$ to obtain $\Sigma^1 = \{\{a, \neg b, \neg c\}, \{b\}, \{c\}\}$. We can again apply unit propagation to Σ^1 on $\{b\}$ to obtain $\Sigma^2 = \{\{a, \neg c\}, \{c\}\}$. Finally, we can apply unit propagation to Σ^2 on $\{c\}$ and then to $\{a\}$ obtain $\Sigma^4 = \{\}$, which is satisfiable. This yields a model $M = \{d, b, c, a\}$ of Σ.

The DPLL procedure is similar to the traditional constraint propagation procedure. The splitting rule in DPLL is similar to the backtracking step in constraint propagation, while the unit propagation rule is similar to the unavoidable steps of consistency checking and forward propagation in traditional CP.

[16] `http://www.satcompetition.org/`

Phase Transitions

We saw that in the worst case, DPLL requires exponential time. Couldn't we do better in the average case? For CNF-formulæ in which the probability for a positive appearance, negative appearance and non-appearance in a clause is 1/3, DP needs on average quadratic time [Gol79]. In retrospect, it was discovered that the formulæ in [Gol79] have a very high probability of being satisfiable. Thus, these formulæ are not representative of those encountered in practice. The idea of *phase transition* was conjectured by [CKT91] to identify hard to solve problem instances:

> *All NP-complete problems have at least one order parameter and the hard to solve problems are around a critical value of this order parameter. This critical value (a phase transition) separates one region from another, such as over-constrained and under-constrained regions of the problem space.*

This conjecture was initially confirmed for the graph coloring and Hamilton path problems and later for other NP-complete problems, including SAT. In the case of SAT problem, an example of the order parameter is ratio of the number of variables to the number of clauses.

1. For higher settings of the parameter, the problem is over-constrained (the formulæ are unsatisfiable). If the probability of a solution is close to 0, this fact can usually be determined early in the search.

2. For lower settings of the parameter, the problem is under-constrained (the formulæ are easily satisfiable). When the probability of a solution is close to 1, there are many solutions, and the first search path of a backtracking search is usually successful.

When this parameter is varied, the problem moves from the over-constrained to the under-constrained region (or vice versa). At phase the transition points, half of the problems are satisfiable and half are not. It is typically in this region that algorithms have difficulty in solving the problem[17]. Cook and Mitchell [CM97] empirically found that for a 3-SAT problem, the phase transition occurs at a clause:variables ratio of around 4.3 (in this experiment, clauses were generated by choosing variables for a clause and complementing each variable with probability 0.5). As an illustration, in the 2003 version of the SAT competition, the largest instances solved using greedy SAT solvers consisted of $100,000$ variables and $1,000,000$ clauses (clause:variable ratio of 10), whereas the smallest unsolved instances comprised 200 variables and $1,000$ clauses (clause:variable ratio of 5). It was also reported in [CM97] that the runtime for the DPLL procedure peaks at the phase transition. In the phase transition region, the DPLL algorithm often near successes. Many benchmark problems are located in the phase transition region, though they have a special structure in addition.

[17] Note that hard instances can also exist in regions of the more easily satisfiable or unsatisfiable instances.

1.3.6 Local Search Methods

Local search methods are standard search procedures for optimization problems. A local search method explores the neighborhood of the current solution and tries to enhance the solution till it cannot do any better. The hope is to produce better configurations through local modifications. The value of a configuration in a logical problem could be measured using the number of satisfied constraints/clauses. However, for logical problems, local maxima are inappropriate; it is required to satisfy all clauses in the theory and not just some. But through random restarts or by noise injection, local maxima can be escaped. In practice, local search performs quite well for finding satisfying assignments of CNF formulæ, especially for under-constrained or over-constrained SAT problems.

GSAT and WalkSat [SKC93] are two local search algorithms to solve boolean satisfiability problems in CNF. They start by assigning a random value to each variable. If the assignment satisfies all clauses, the algorithm terminates, returning the assignment. Otherwise, an unsatified clause is selected and the value of exactly one variable changed. Due to the conjunctive normal form, flipping one variable will result in that clause becoming satisfied. The above is then repeated until all the clauses are satisfied. WalkSAT and GSAT differ in the methods used to select the variable to flip. While GSAT makes the change which minimizes the number of unsatisfied clauses in the new assignment, WalkSAT selects the variable that, when flipped, results in no previously satified clauses becoming unsatified (some sort of downward compatibility requirement). MaxWalkSat is a variant of WalkSat designed to solve the weighted satisfiability problem, in which each clause is associated with a weight. The goal in MaxWalkSat is to find an assignment (which may or may not satisfy the entire formula) that maximizes the total weight of the clauses satisfied by that assignment. These algorithms perform very well on the randomly generated formulæ in the phase transition region. Monitoring the search procedure of these greedy solvers reveals that in the begginning, each procedure is very good at reducing the number of unsatisfied clauses. However, it takes a long time to satisfy the few remaining clauses (called plateaus). The GSAT algorithm is outlined in Figure 1.9.

1.3.7 Default inference under closed world assumption

Given any set of formulae Σ, the closed-world assumption is the assumption that Σ determines all the knowledge there is to be had about the formulae in the language. Thus, if we consider any proposition[18] A then A is taken to be true exactly when Σ (logically) implies A, but is otherwise taken to be false.

The closed-world assumption underlies the mode of reasoning known as *default inference*. There are many situations, both in ordinary daily life and in specific technical computing matters (such as explaining the theory of finite failure and the relationship it bears to reasoning with negative information), when

[18] In the first order logic programming context, the propositions in which we are primarily interested are the atoms of the Herbrand base.

INPUT: A set of clauses Σ, MAX-FLIPS, and MAX-TRIES.
OUTPUT: A satisfying truth assignment of Σ, if found.
for $i = 1$ to MAX-TRIES **do**
  T = a randomly-generated truth assignment.
  for j:=1 to MAX-FLIPS **do**
    if T satisfies Σ **then**
      return T
    end if
    v = a propositional variable such that a change in its truth assignment gives the largest increase in the number of clauses of Σ that are satisfied by T.
    $T = T$ with the truth assignment of v reversed.
  end for
end for
return "Unsatisfiable".

Figure 1.9: Procedure GSAT.

default inference is a necessary supplement to deductive inference. Consider this simple example of a single clause axiom $\Sigma = \{A \leftarrow B\}$ for which $B(\Sigma)$ is just $\{A, B\}$. Under the closed-world assumption, we may infer $\neg A$ for any ground atom $A \in B(\Sigma)$ that us not implied by Σ. This is a constrained[19] application of the rule of default inference:

Infer $\neg A$ in default of Σ implying A

Motivated by the desire to draw sound conclusions about negative information, we will consider two constructions that provide consequence-oriented meaning for default inference under the closed-world assumption.

1. CWA(Σ) : The combination of Σ with the default conclusions inferred from it is denoted by CWA(Σ) and is defined by

$$\text{CWA}(\Sigma) = \Sigma \cup \{\neg A \mid A \in B(\Sigma) \textit{ and not } \Sigma \models A\}$$

For the above example, $\text{CWA}(\Sigma) = \{A \leftarrow B, \neg A, \neg B\}$.

Soundness for the default inference of negative conclusions under the closed-world assumption can be referred to the the logical construction CWA(Σ) as follows:

$$for\ all\ A \in B(\Sigma),\ \text{CWA}(\Sigma) \models \neg A\ if\ \Sigma \vdash_{CWA} \neg A$$

There some practical problems with CWA:

[19] Constrained because (i) Σ is assumed to be inconsistent, else Σ would necessarily imply both A and $\neg A$ (ii) only the case where A is atomic is considered, otherwise if Σ implied, say, neither A nor $\neg A$, then the default rule would infer both $\neg A$ and $\neg A$, which would again be inconsistent.

(a) One problem is that, we have no immediate way of writing down CWA(Σ), for we cannot directly discern which particular negative facts $\neg A$ to include in it; we would have to infer them all first.

(b) A more serious defect is that, whilst CWA(Σ) is always consistent when Σ is definite[20], it is likely to be inconsistent otherwise. For example, if Σ comprises just the clause $\{A \vee B\}$, then CWA(Σ) = $\{A \vee B, \neg A, \neg B\}$ which is inconsistent.

2. CWA(Σ) : For default inference applied to indefinite programs, we refer the soundness criterion to *completion* COMP(Σ). It is also known as the completed database of Σ. Unlike CWA(Σ), it can be written down more-or-less directly and is consistent for all well-structured programs. In the case of propositional logic, the construction is particularly simple:

   (a) Initialize COMP(Σ) = $\emptyset$.

   (b) Assume that Σ is any set of clauses of the form ($A \leftarrow$ *body*).

   (c) For each A mentioned in Σ but not defined in Σ, construct $\neg A$ and add it to COMP(Σ).

   (d) For each A having a definition in Σ of the form

   $$A \leftarrow \ body - 1$$
   $$\ldots$$
   $$\ldots$$
   $$A \leftarrow \ body - n$$

   construct the clause A iff ($body - 1 \vee \ldots \vee body - n$) and add it to COMP($\Sigma$),

Comp(Σ) can also be viewed as the result of simplifying $\Sigma \cup$ only-if(Σ), where only-if(Σ) comprises every completed definition of the form $\neg A$ for $A \notin \Sigma$ as well as ($q \rightarrow \alpha$) for every completed definition (q iff α) $\in$ COMP(Σ).

Following are some characteristics of completions

(a) In general it is also a more economical construction in the sense that it implies, but does not necessarily declare, various negative propositions which CWA(Σ) would have to declare explicitly. As an example, if $\Sigma = \{A \leftarrow B\}$, COMP($\Sigma$) = $\{\neg B, A$ iff $B\}$. While Σ neither implies A nor implies B, *Comp*(P) implies both $\neg A$ and $\neg B$, which is exactly the same outcome obtained using CWA(Σ) instead. Observe, however, that CWA(Σ) declares $\neg A$ **explicitly** whereas *Comp*(Σ) does not.

[20] Exercise: Prove.

(b) Generally, completion is more conservative than the closed-world assumption in the negative facts that it implies. For example, with $\Sigma_1 = \{A \leftarrow A\}$, $\text{CWA}(\Sigma_1) \models \neg A$, whereas $\text{COMP}(\Sigma_1) \not\models \neg A$. Similarly, with $\Sigma_2 = \{A \leftarrow \neg B\}$, $\text{CWA}(\Sigma_2) \models \neg A \wedge \neg B$ (is inconsistent), whereas $\text{COMP}(\Sigma_2) \not\models \neg A$ though $\text{COMP}(\Sigma_2) \models \neg B$.

(c) On the other hand, COMP can be sometimes less conservative; for $\Sigma_3 = \{A \leftarrow \neg A\}$, $\text{COMP}(\Sigma_3) \models \textit{everything}$, whereas $\text{CWA}(\Sigma_3) \models A$ and $\text{CWA}(\Sigma_3) \not\models \neg A$.

(d) Completed definitions capture the programmer's intentions more fully than do uncompleted definitions in the original program. For computational purposes however, the program alone happens to be sufficient for deducing all intended answers.

(e) In cases where Σ is indefinite, the elementary consequences of $\text{COMP}(\Sigma)$ may arise from the joint contributions of Σ and only-if(Σ) and not from either of them alone.

(f) There is a syntactic bias built into the process of program completion. For example, if $\Sigma_1 = \{A \leftarrow \neg B\}$ and $\Sigma_2 = B \leftarrow \neg A$ their respective completions are $\text{COMP}(\Sigma_1) = \{A \text{ iff } \neg B, \neg B\}$ and $\text{COMP}(\Sigma_2) = \{B \text{ iff } \neg A, \neg A\}$. These completions are not logically equivalent despite the fact that the original programs are. The difference between their completions deliberately reflects the difference between the procedural intentions suggested by the programs' syntaxes: the first program anticipates queries of the form $?A$ whilst the second anticipates queries of the form $?B$.

Finite Failure Extension

We look at one further extension of SLD-resolution that is of special interest to us, namely, the idea of negation as "failure to prove". **We know that SLD alone is able to deal soundly and completely with all positive atomic queries A posed to this database. But with the finite failure facility we can now also ask directly, in our extended language, whether some atom is *not* in the database.**

This extension enables one to express, via a call '*fail* A', the condition that a call A shall finitely fail. In Prolog the fail operator is denoted by '*not*' and is referred to as 'negation by failure'. The operational meaning of '*fail*' is summed up by the finite failure rule:

> A call '*fail* A' succeeds *iff* its subcall A finitely fails.

This rule can be incorporated directly into the execution strategy for virtually no cost in terms of implementation overheads - to evaluate a call '*fail* A', the interpreter merely evaluates A in the standard way and then responds to the outcome as just prescribed. The phrase 'A finitely fails' means that the execution tree generated by the evaluation of A must be a finitely failed tree -

that is, it must have finite depth, finite breadth and all the computations contained within it must terminate with failure. By contrast, should one or more of those computations terminate with success then the call '*fail* A' is itself deemed to have finitely failed. Finally, if the evaluation of of A neither succeeds nor finitely fails, then the same holds for the evaluation of '*fail* A'

The rule of "negation as finite failure" states that if all branches of an SLD-tree finitely fail for $\Sigma \cup \{\leftarrow A\}$ for a set of definite clauses Σ and a ground literal A, we can derive the ground literal $\neg A$ as a result. This combination of SLD-resolution and negation-as-failure results in the proof strategy we called SLDNF-resolution. Once again, let us look at an example:

C_0 : $\neg$ Fred is an ape

C_1 : Fred is an ape $\leftarrow$ *not* Fred is human, Fred is a primate

C_3 : Fred is a primate

Here, *not* stands for "not provable" (which is not the same as $\neg$). C_0 and C_1 resolve to give R_1 : $\leftarrow$ *not* Fred is human, Fred is a primate. With a rightmost literal computation rule as before, the next resolvent is R_2 : $\leftarrow$ *not* Fred is human. It is evident that Fred is human is not provable and $\square$ results.

SLDNF is sound and complete for propositional definite clauses. However, there are some important issues in extending SLD with finite failure to first order logic such as (i) incompleteness when applied to activated non-ground fail calls and (ii) unsoundness in certain cases. To solve the latter, more serious problem, a selection policy called *safe computation rule* is used in practice; this policy sacrifices completeness for the sake of soundness. These and other concepts such as *floundering* will be discussed in a later section.

The SLD finite failure set

Relative to a given Σ and a given inference system R, the SLD finite failure $FF(\Sigma, R)$ set comprises exactly propositional atoms for which the query $?q$ finitely fails. We shall restrict our attention throughout to the case where Σ is definite and R is SLD. Once we know what the finite failure set is relative to SLD inference, we shall also be able to say something about the execution by SLDNF of queries containing 'fail calls' - subject, of course, to the assumption that the chosen computation rule is safe. The more general situation where 'fail' calls may occur also in clause bodies, is significantly more complicated, and will not be addressed here. For queries of the form $?q$ using a definite Σ under some SLD computation rule R partitions $B(\Sigma)$ into three species of atoms

1. $SS(\Sigma, R)$, or those for which $?q$ succeeds. Since, if $?q$ succeeded in one SLD tree, it succeeded in all SLD trees, the capacity for $?q$ to succeed is independent of the computation rule. So this set can be simply denoted by $SS(\Sigma)$.

2. $FF(\Sigma, R)$, or those for which $?q$ finitely fails

3. $IF(\Sigma, R)$, or those for which $?q$ infinitely fails.

However, $?q$ may finitely fail under one computation rule yet infinitely fail under another. A simple example is where Σ comprises just the clause $q \leftarrow B \wedge q$. Under the standard leftmost call rule, the query $?q$ finitely fails, whereas it infinitely fails under a rule which always selects the rightmost call. Thus the boundary between $FF(\Sigma, R)$ and $IF(\Sigma, R)$ depends upon R. There is a simple theoretical way of eliminating this dependence upon R. We invoke the idea of a particular sort of computation rule which ensures that any call introduced into a computation is selected after some arbitrary but finite number of execution steps. Such a rule is called a *fair computation rule*. Considering the example again, we might allow a fair computation rule to select 'q' calls any finite number of times, but sooner or later the fairness requirement would demand that an 'A' call be selected, thus immediately forcing finite failure. With this new concept, we can now state the following facts: $?q$ finitely fails under some computation rule *iff* it finitely fails under all fair computation rules[21] We can denote by $FF(\Sigma)$, the set of all atoms q for which $?q$ has some finitely failed fair SLD-tree, and by $IF(\Sigma)$, the set of all atoms q for which $?q$ has some infinitely failed fair SLD-tree. $FF(\Sigma)$ is called the (fair-) SLD finite failure set of Σ. Referring again to the example above, we shall have $FF(\Sigma) = \{q, A\}$ and $IF(\Sigma) = \emptyset$.

Suppose queries were also allowed to contain '*fail*' calls besides atomic ones, but with Σ still restricted to be definite. The following statements then hold true:

$$\begin{array}{lllll} \text{for all } q \in B(\Sigma), & q \in FF(\Sigma) & \text{iff} & ?fail\ q \text{ succeeds under SLDNF} \\ & q \in SS(\Sigma) & \text{iff} & ?q \text{ succeeds under SLDNF} \end{array}$$

We can interpret finite failure (i) in terms of the position of $FF(\Sigma)$ within the lattice of interpretations and (ii) in terms of the classical negation ($\neg$), which provides a semantics for '*fail*', and some soundness and completeness results for SLDNF, based upon the logical consequences of COMP(Σ).

Completion Semantics for SLDNF

For pure Horn-clause programs and queries, we had a particularly simple connection between logical meaning and operational meaning:

$$\text{for all } q \in B(\Sigma),\ \Sigma \models q \textit{ iff } q \in SS(\Sigma)$$

The construction of a consequence-oriented semantics for the finite failure extension is more problematic. A program containing fail is not a construct of classical logic and so is not amenable to the notion of classical logical consequence. Nevertheless, a variety of analogous connections have been suggested. The best-known of these is based upon the so-called *completion semantics*, which relies upon two ideas:

[21] Although fair computation rules are not normally implemented, they certainly tidy up our mathematical account of finite failure.; equivalently, at least one of its SLD-trees is finitely failed *iff* all of its fair SLD-trees are finitely failed.

1. Interpreting 'fail' as the classical negation connective $\neg$ (this is why fail is commonly referred to as 'negation by failure').

2. Relating the success or finite failure of calls to logical consequences of COMP(Σ) rather than of Σ alone.

On this basis we can then characterize, in logical terms, the *soundness* of execution of atomic queries under the (safe-)SLDNF implementation of fail:

for all $q \in B(\Sigma)$,	COMP(Σ) $\models q$	if $?q$	succeeds under SLDNF
	COMP(Σ) $\models \neg q$	if $?\ fail\ q$	succeeds under SLDNF

These results hold even if '*fai;*' calls occur in Σ. The 'only-if' part in the two statements above, which characterizes the *completeness* for SLDNF holds only when Σ is definite (so Σ will obviously not contain 'fail' calls).

There are two caveats in the above statement(s): (i) COMP(Σ) may itself be inconsistent and (ii) the SLDNF uses some safe computation rule. Also, the simplification of analysis of finite failure using *fair* computation rules is not possible when the query contains '*fail*' calls.

1.3.8 Lattice of Models

For any Σ that is a set of definite clausal formula, it can be shown that the set $\mathcal{M}(\Sigma)$ of models is a complete lattice ordered by set-inclusion (that is, for models $M1, M2 \in \mathcal{M}(\Sigma)$, $M1 \preceq M2$ 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 thar 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 models for the formula. We will illustrate with an example. Consider a Σ consisting of the following clauses:

$$C_0 : \text{CC} \leftarrow \text{CL}$$
$$C_1 : \text{CB} \leftarrow \neg\ \text{BL}$$
$$C_2 : \text{CL} \leftarrow \text{LL}$$
$$C_3 : \text{BL}$$

A fail-safe way of getting a model is to choose the entire set of propositions $B(\Sigma)$ ={CC, CL, CB, LC, LL, LB, BC, BL, BB}, which is called the *base* of Σ. for then every atom in Σ is assigned true and this in turn makes all of its clauses true. This model is the (unique) maximal model for Σ. However, such a choice is clearly excessive-many of the atoms in B(P) do not even occur in G(P) and so their truth values are irrelevant. Stripping out those irrelevant atoms leaves a somewhat smaller model {CC, CL, CB, BL, LL}. Which of these atoms must appear in any model? Clearly BL must, in order to make the program's fourth

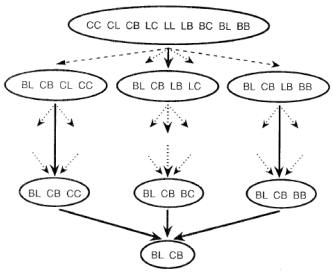


Figure 1.10: A complete lattice of models.

clause true. Then, since BL must, so also must CB in order to make the second clause true. The first and third clauses employ only the remaining three atoms CC, CL and LL, and both those clauses can he made true by making those atoms false. In conclusion, only BL and CB *must* be true – thus the (unique) minimal model for Σ is {BL,CB}.

Some indication of the complete lattice of models for the program is shown in Figure 1.10 where, edges stand for the *covers* relationship. Moreover, only a few of the models are shown-there are 64 models in the entire lattice. Note also that, in general, most subsets of $B(\Sigma)$ will be counter-models–for our example there are 448 of these, of which the smallest is $\emptyset$. It can be proved that if Σ is a set of definite clauses, it must always be satisfiable, and therefore, $\emptyset \notin \mathcal{M}(\Sigma)$

The reason why the discussion above has focused solely upon definite clauses is that, in general, a set of indefinite clauses does not yield a complete lattice and may therefore have multiple minimal models. The lack of a unique minimal model makes it harder to assign an unambiguous meaning to such a set.

There is an important result relating a set of definite clausal formulae Σ, its minimal model $\mathcal{MM}(\Sigma)$ and the atoms that are logical consequences of Σ:

Theorem 12 If α is an atom then $\Sigma \models \alpha$ if and only if $\alpha \in MM(\Sigma)$.

The proof of theorem 12 makes use of the so-called model intersection property:

Theorem 13 *If $M_1, \ldots, M_n$ are any models for a definite clause set Σ then their intersection is a model of Σ.*

Proof: It is easy to show this by induction. For any $k < n$, let I_k denote $M_1 \cap \ldots \cap M_k$. Now consider any clause $C \in \Sigma$:

C : $A \leftarrow B_1 \wedge \ldots \wedge B_m$

We shall prove that, for all $k \leq n$, I_k satisfies C.

1. *Base case ($k = 1$):* $I_1 = M_1$ and therefore satisfies C.
2. *Induction step ($1 \leq k \leq n$)*: Assume that M_k satisfies C;

if I_k satisfies C then	either	$B_i \notin I_k$ for some i
	or	$B_i \in I_k$ for all i, and $A \in I_k$
if M_{k+1} satisfies C then	either	$B_i \notin M_{k+1}$ for some i
	or	$B_i \in M_{k+1}$ for all i, and $A \in M_{k+1}$
it then follows that	either	$B_i \notin I_k \cap M_{k+1}$ for some i
	OR	$B_i \in I_k \cap M_{k+1}$ for all i, and $A \in I_k \cap M_{k+1}$
And hence		I_{k+1} satisfies C

Thus for all $k \leq n$, I_k satisfies C and-by a similar argument – every other clause in Σ.

□

We next prove theorem 12 making use of theorem 13.

Proof of theore 12: EXERCISE

Using this result it is now easy to prove the relationship between $MM(\Sigma)$ and any atomic consequence q of Σ:

If $\Sigma \models q$	then	q is true in every model of Σ
	then	$q \in I^*$
	then	$q \in MM(\Sigma)$
if $q \in MM(\Sigma)$	then	q is true in every model of Σ
	then	$\Sigma \models q$

□

Thus, the minimal model of a definite clausal formula is identical to the set of all ground atoms logically implied by that formula, which was defined as the success set $SS(\Sigma)$. Thus, the minimal model provides, in effect, the meaning (or semantics) of the formula. We shall see later that this is just one of several ways of giving significance to a program's minimal model.

We can envisage a procedure for enumerating the models of a formula. Consider the powerset of the base $B(\Sigma)$. 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 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 base. Suppose now we start at some point s in this sublattice, and we move to a new point that consists only of those 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 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_Σ that, for a set of clauses Σ, generates an interpretation (not necessarily a model) from another. That is:

$$I_{k+1} = T_\Sigma(I_k)$$

where

$$T_\Sigma(I) = \{a : a \leftarrow body \in \Sigma \text{ and } body \in I\}$$

It can be shown that T_Σ is both monotonic and continuous on the complete lattice obtained by ordering the powerset of the base by $\subseteq$. So, we know from the Knaster-Tarski Theorem mentioned on page 11, that there must be a least fixpoint for T_Σ in this lattice. We can prove that the procedure of obtaining I_{k+1} from application of T_Σ to I_k will yield that fixpoint, and further, that this fixpoint will be the minimal model.

Theorem 14 *$MM(\Sigma)$ is the least fixpoint of T_Σ.*

Proof: The definition of T_Σ requires that

for all $I \subseteq B(\Sigma)$ and for all q, $q \in T_\Sigma(I)$ iff $(\exists body)[(q \leftarrow body) \in \Sigma$ and $body \in I]$

whereas, the definition of a model requires that

for all $I \subseteq B(\Sigma)$, I is a model for Σ iff for all q, $q \in I$ if $(\exists body)[(q \leftarrow body) \in \Sigma$ and $body \in I]$.

These two sentences jointly imply

$$\begin{array}{lll} \text{for all } I \subseteq B(\Sigma) & & \\ I \text{ is a model for } \Sigma & \text{iff} & \text{for all } q,\ q \in I \text{ if } q \in T_\Sigma(I) \\ & \text{iff} & T_\Sigma(I) \subseteq I \\ & \text{iff} & I \text{ is a pre-fixpoint of } T_\Sigma \end{array}$$

Where, for a function f on $< S, \preceq >$, an element $u \in S$ is a pre-fixpoint of f if and only if $f(u) \preceq u$. Then, we can recall from the proof of the Knaster-Tarski theorem on page 11 that the least fix point is also the least pre-fixpoint. Hence, since the models are exactly the pre-fixpoints, the least model $MM(\Sigma)$, which is the glb of all the pre-fix points, must be the least pre-fixpoint. Finally, by the Knaster-Tarski theorem, $MM(\Sigma)$ must also be the least fixpoint. □

So we now have several equivalent characterizations of the minimal model, including the success set $SS(\Sigma)$ and the new one posed in terms of the least fixpoint of T_Σ, denoted $LFP(T_\Sigma)$

Mathematical Characterization of Finite Failure

It so happens that there is a somewhat related (to the T_Σ function) method of constructing the finite failure set $FF(\Sigma)$. The set of atoms which occur as headings in Σ is simply $T_\Sigma(B(\Sigma))$, as is plain from the definition of T_Σ. The simplest way for $?q$ to fail finitely is for there to be no clause in Σ who heading

unifies with q. In this case the failure is said to occur within depth $k = 1$ and the set of all such atoms $q \in B(\Sigma)$ is denoted by $FF(\Sigma, 1)$ and can be characterized very easily using the T_Σ function as

$$FF(\Sigma, 1) = B(\Sigma) - T_\Sigma(B(\Sigma))$$

Generalizing this principle, $FF(\Sigma)$ just contains each atom q for which $?q$ finitely fails within some depth $k \in \mathcal{N}$.

$$FF(\Sigma) = \cup_{k \in \mathcal{N}} FF(\Sigma, k) = B(\Sigma) - \cap_{k \in \mathcal{N}} T_\Sigma^k(B(\Sigma))$$

When we start at the top element $B(\Sigma)$ of our lattice of interpretations, repeated application of the T_Σ function generates a monotonically decreasing sequence $B(\Sigma) \supseteq T_\Sigma(B(\Sigma)) \supseteq T_\Sigma^2(B(\Sigma)) \ldots$, whose limit is the greatest lower bound (glb) $T_\Sigma \downarrow$ of $\left\{T_\Sigma^k(B(\Sigma)) \mid k \in \mathcal{N}\right\}$. Thus, the mathematical characterization of $FF(\Sigma)$ (independent of the execution mechanism) is

$$FF(\Sigma) = B(\Sigma) - T_\Sigma \downarrow$$

Based on this equivalence, what is the value of $FF(\Sigma)$ for the last example that we discussed?

It can be proved that $T_\Sigma \uparrow = LFP(T_\Sigma) = MM(\Sigma) \subseteq T_\Sigma \downarrow$. There is an assymetry in the relationship between the limits and the extremal fixpoints of T_Σ. Whereas, $T_\Sigma \uparrow$ is always equal to the least fix point $LFP(T_\Sigma)$, $T_\Sigma \downarrow$ does not always equal the greatest fix point $GFP(T_\Sigma)$, though it does hold for most 'sensible' Σ (at the least for non-recursive clauses and in the case of 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.