Logic and Automating Proof

Jane Hesketh

November 1, 1988

This is not meant to be a complete introduction to logic. Just some scene setting and notes on particular points. Some of the notes on the use of Gentzen Sequent Calculus follow Gallier's book "Logic for Computer Science".

1 Propositional Calculus

The propositional calculus is a language for labelling events and combinations of events which may be true or false, and then ascertaining the truth or falsity of the combinations. The statements in the language are considered relative to *interpretations*, assignments of truth and falsity to the propositions which reflect our understanding of what we intend the symbols to mean.

1.1 Syntax

The calculus consists of:

- atomic propositions which we may label as we like: *it is raining, there is nothing good on TV tonight*, etc.
- connectives: $\wedge, \vee, \supset, \neg, \equiv$, respectively and, or, implies, not and equivalence. Sometimes other symbols are used instead of these.
- Well-formed formulae built up from these.
- Also used sometimes are symbols for false and true: $\bot, \top$. Brackets are often used for clarity or to require precedence of certain connectives over others.

Not all combinations of symbols are legitimate formulae which can be evaluated. Well-formed formulae are :

- Propositions
- $\neg P, P \wedge Q, P \vee Q, P \supset Q, P \equiv Q$ where P and Q are both well-formed formulae.
- Nothing Else

I will use letters like P and Q as variables, representing any formula.

Some connectives are deemed to bind tighter than others, from tightest to loosest they are:

- $\neg$
- $\vee, \wedge$
- $\supset, \equiv$

Note that for the purpose of description, it is often useful to use propositional variables, i.e. letters which represent some atomic proposition.

1.2 Truth Values of Formulae

Propositions may be assigned either the value *true* ($\top$) or the value *false* ($\bot$). The truth value of formulae are established recursively from the truth values of the argument(s) of the dominant connective. So for some formula of the form $A \vee B$, it will be true if A is true or if B is true. A and B may in turn be formulae.

The following tables show the truth value to be assigned to the overall formula according to the truth values of the argument(s). (Vertical is left and horizontal is right).

$\wedge$	$\top$	$\bot$
$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

$\vee$	$\top$	$\bot$
$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

$\supset$	$\top$	$\bot$
$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

$T \rightarrow F \models F.$

$\equiv$	$\top$	$\bot$
$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\top$

$\neg$	$\top$	$\bot$
	$\bot$	$\top$

1.3 Truth Values of Formulae

In general it is quite easy to work out the possible truth values of a formula for all possible assignments of truth and falsity to the propositions in it. Each proposition

named in the formula may be true or false, so if there are 2 propositions, there are 2^2 possible assignments of truth values, and for 3 variables 2^3 possible assignments etc.

Two common ways of doing this are truth tables and semantic trees.

1.3.1 Truth Tables

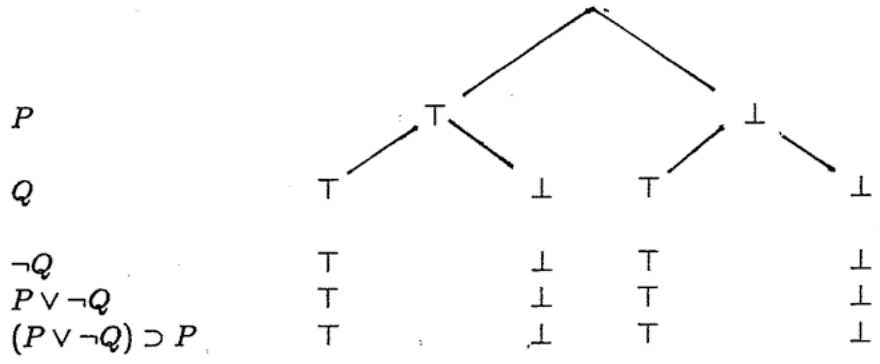
Each column contains truth values for a formula depending on the different truth assignments to the propositions occurring in it. On the left the formulae are propositions, and to the right, increasingly more complicated formulae built from the formulae to the left.

Each row contains the truth assignments to the various formulae depending on the truth assignments to the propositions. So if there are n different propositions, the leftmost n columns record their combinations of true/false assignments, and there are 2^n rows.

P	Q	$\neg Q$	$P \vee \neg Q$	$(P \vee \neg Q) \supset P$
$\top$	$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\bot$	$\bot$	$\top$	$\top$	$\bot$

1.3.2 Semantic Trees

This is much the same sort of idea as truth tables. The tree is as deep as there are propositions. Each level corresponds to assigning its proposition true or false for each of the assignments already made to other variables. So each path from the root to a leaf is a unique assignment of truth values to the propositions.



1.4 Equivalences and Redundancies

Using such methods, you can see that some connectives and formulae have exactly the same truth functional properties as other formulae. So some formulae are precisely equivalent to others, and may be eliminated in favour of the others. This can be used to reduce the number of connectives present in a formula. (In fact there are connectives such as NAND, which I haven't chosen to use here, which can express all the others when used in the right combinations).

P	Q	$\neg P$	$\neg\neg P$	$\neg P \vee Q$	$P \supset Q$	$Q \supset P$	$P \supset Q \wedge Q \supset P$	$P \equiv Q$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$

Look at the columns for P and $\neg\neg P$ to see their logical equivalence. Similarly $P \supset Q$ and $\neg P \vee Q$, $P \equiv Q$ and $(P \supset Q) \wedge (Q \supset P)$.

Other equivalences:

$(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$

$(P \vee Q) \wedge R \equiv (P \wedge R) \vee (Q \wedge R)$

1.5 Satisfiablity, Falsifiability and Theorems

Using devices such as truth tables and semantic trees, it is straightforward to work out whether some formula is:

- Always true, regardless of truth assignments to its propositions - it is are true in all interpretations. This is called a *tautology* – these are the theorems of the calculus. e.g. $P \vee \neg P$
- Never true, regardless of truth assignments to its propositions - it is false in all interpretations. This is *unsatisfiable*. e.g. $P \wedge \neg P$
- True for some assignment of truth values. It is *satisfiable*.
- False for some assignment of truth values. It is *falsifiable*.

Naturally these formulae with the special property of always being true are very interesting, and the ones we want to know about.

1.6 Proof

Although it is possible to consider all possible 2^n assignments of truth to all propositions, the number of these increases exponentially with the number of propositions. Clearly it is nicer to use our understanding of what a tautology is to avoid such expensive enumeration. There are various possibilities such as

- using conjunctive normal form
- searching for falsifiability, in which case it cannot be a tautology, and at least you can stop as soon as you find this.
- splitting the problem up into more manageable components.

1.6.1 Conjunctive Normal Form

A conjunction of disjunctions of literals. (A literal is an atomic propostion or the negation of one). E.g.

$$(P \vee Q \vee \neg R) \wedge (S \vee \neg Q) \wedge \neg P$$

A conjunction is a tautology if and only all its conjuncts are tautologies. Such a disjunction is a tautology if it contains an atomic proposition and its negation amongst its disjuncts. This is easy to detect, just check all the disjuncts. If any is falsifiable, the whole expression is falsifiable.

To convert a formula to CNF:

1. Eliminate $\equiv$ and $\supset$ in favour of $\wedge$, $\vee$ and $\neg$
2. Drive all $\neg$ down to propositions and eliminate $\neg\neg$
3. Use distributivity of $\wedge$ and $\vee$

E.g.

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

However all this still involves the overhead of normal forming. Using a slightly different logical system, normal forming can be avoided. The next subsection will describe such a system for propositions, and algorithms for searching in it.

1.7 Gentzen Sequent Calculus

A *sequent* is written as

$$A_1, ..., A_m \rightarrow B_1, ..., B_n$$

and corresponds to

$$(A_1 \wedge ... \wedge A_m) \supset (B_1 \vee ... \vee B_n)$$

$\rightarrow$ corresponds to a single top level implication. The left-hand list of propositional formulae are considered to be ANDed together, and the right-hand list are considered to be ORed together. So for any sequent:

- tautology means having the same formula on both sides of $\rightarrow$ e.g. $P \rightarrow P, Q$. This is called an *axiom* in this context. It is obviously not falsifiable.

- falsifiablity requires that *all* the formulae on the left-hand-side can be made true and simultaneously one of the formulae on the right-hand-side can be made false. E.g $P \rightarrow T$

The left-hand-side is referred to as the antecedent, and the right-hand-side as the succedent. You can think of these as assumptions and conclusions respectively.

Either the succedent or the antecedent may be empty. If m is zero, i.e. the antecedent is empty, that corresponds to no assumptions being required to prove the succedent. If both antecedent and succedent are empty, that is the inconsistent sequent, always false.

Proof is investigated using deduction trees like this one, which demonstrates that $(P \supset Q) \supset (\neg Q \supset \neg P)$ is a tautology :

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg Q, P \rightarrow P}{\neg Q \rightarrow P, \neg P}}{\rightarrow P, \neg Q \supset \neg P} \quad \dfrac{\dfrac{Q \rightarrow \neg P, Q}{Q, \neg Q \rightarrow \neg P}}{Q \rightarrow \neg Q \supset \neg P}}{P \supset Q \rightarrow \neg Q \supset \neg P}}{\rightarrow (P \supset Q) \supset (\neg Q \supset \neg P)}$$

This tree demonstrates the validity of $(P \supset Q) \supset (\neg Q \supset \neg P)$ because it is validly constructed according to the inference rules (see below) and each leaf is an *axiom*. So, it starts with valid sequents and each sequent is validly inferred from the one(s) above it. Gallier's book contains proofs that these trees are equivalent to truth tables in their demonstration of validity and falsifiability, and that this procedure is sound and complete. It is important to know that these properties can be proved.

The obvious way to grow a deduction tree is from the bottom. The bottom sequent here can be thought of as the statement that no assumptions are required to prove $(P \supset Q) \supset (\neg Q \supset \neg P)$. Then use the inference rules backwards, to grow sequents from which the ones below could have been derived. For each sequent there will be as many ways of doing this as there are non-atomic formulae in the sequent. You can view the growth of the tree as a search for all leaves being axioms, but if at any point you find a non-axiom, you know the sequent is falsifiable, and so is everything below derived from it. So there is a potential for doing less work than a complete normal forming, since you may be able to stop short if you find a falsifiable sequent. It depends on how you grow the tree. There are choices - breadth-first, depth-first, preferring one-premise rules to two-premise rules, to delay branching as long as possible....

1.7.1 Inference Rules

$$\frac{\Gamma, A, B, \Delta \rightarrow \Lambda}{\Gamma, A \wedge B, \Delta \rightarrow \Lambda} \qquad \frac{\Gamma \rightarrow \Delta, A, \Lambda \qquad \Gamma \rightarrow \Delta, B, \Lambda}{\Gamma \rightarrow \Delta, A \wedge B, \Lambda}$$

$$\frac{\Gamma, A, \Delta \rightarrow \Lambda \qquad \Gamma, B, \Delta \rightarrow \Lambda}{\Gamma, A \vee B, \Delta \rightarrow \Lambda} \qquad \frac{\Gamma \rightarrow \Delta, A, B, \Lambda}{\Gamma \rightarrow \Delta, A \vee B, \Lambda}$$

$$\frac{\Gamma, \Delta \rightarrow A, \Lambda \qquad B, \Gamma, \Delta \rightarrow \Lambda}{\Gamma, A \supset B, \Delta \rightarrow \Lambda} \qquad \frac{A, \Gamma \rightarrow B, \Delta, \Lambda}{\Gamma \rightarrow \Delta, A \supset B, \Lambda}$$

$$\frac{\Gamma, \Delta \rightarrow A, \Lambda}{\Gamma, \neg A, \Delta \rightarrow \Lambda} \qquad \frac{A, \Gamma \rightarrow \Delta, \Lambda}{\Gamma \rightarrow \Delta, \neg A, \Lambda}$$

where Γ, Δ and Λ are arbitrary sequences of formulae, and A and B are propositional formulae. Strictly speaking there are also some structural rules which permit you to reorder the formulae on either side, and duplicate formulae for use with different inference rules. Each of these rules is labelled with the name of the connective introduced and the side of the sequent on which it is introduced.

The sequent(s) above the line is the premise(s) and that below the line is the conclusion. Since these rules always reduce the number of connectives when applied backwards, the growth of the tree backwards is guaranteed to terminate.

1.7.2 Tree Exploring Algorithm

This is a sketch of such an algorithm in a Pascal-like language, there are two procedures:

```
procedure search(Γ → Δ:sequent; var T:tree) ;
    begin
        let T be the one-node tree labelled with Γ → Δ ;
        while not all leaves of T are finished do
            T0 := T ;
            for each leaf node of T0 do
                if not finished(node) then
                    expand(node, T)
                endif
            endfor
        endwhile;
        if all leaves are axioms
        then
            write( 'T is a proof of Γ → Δ' )
        else
            write( Γ → Δ is falsifiable' )
        endif
    end
```

```
procedure expand(node:tree-address ; variiT :tree );
    begin
        let A_1,...,A_m → B_1,...,B_n be the label of the node;
        let S be the one-node tree labelled with A_1,...,A_m → B_1,...,B_n;
        for i := 1 to m do
            if non-atomic(A_i) then
                S := the new tree obtained from S by applying to the
                descendant of A_i in every non-axiom leaf of S
                th left rule applicable to A_i;
            endif
        endfor;
        for i := 1 to n do
            if non-atomic(B_i) then
                S := the new tree obtained from S by applying to the
                descendant of B_i in every non-axiom leaf of S
                the right rule applicable to B_i;
            endif
        endfor;
        T:= dosubstitution(T,node,S)
    end
```

That algorithm would produce this tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Q,P,P,S \to Q,T,T}{Q,P,\neg Q,P,S \to T,T}}{Q,P,\neg Q,(P \wedge S) \to T,T} \quad \dfrac{\dfrac{R,Q,P,P,S \to Q,T}{R,Q,P,\neg Q,P,S \to T}}{R,Q,P,\neg Q,(P \wedge S) \to T}}{Q,P,\neg Q,(T \supset R),(P \wedge S) \to T}}{P,\neg Q,(T \supset R),(P \wedge S) \to P,T}}{P,\neg Q,(P \supset Q),(T \supset R),(P \wedge S) \to T}}{(P \wedge \neg Q),(P \supset Q),(T \supset R),(P \wedge S) \to T}$$

which shows that $(P \wedge \neg Q),(P \supset Q),(T \supset R),(P \wedge S) \to T$ is a tautology. It would also produce this tree:

$$\dfrac{\dfrac{P \to Q}{\to P \supset Q} \quad \dfrac{\dfrac{Q \to P}{Q,\neg P \to}}{\neg P \to \neg Q}}{\to (P \supset Q) \wedge (\neg P \supset \neg Q)}$$

which is a counter-example tree. The leaves are not axioms, and the root sequent is not a tautology. So this tree shows that the root sequent is falsifiable.

Gallier's book also contains a proofs that:

- these procedures for developing the tree terminate for every finite input sequent $\Gamma \rightarrow \Delta$. This is done by showing that the number of connectives is constantly being reduced.
- the procedures are complete - if the root sequent is not a tautology, they will generate a tree which falsifies the root sequent.
- the procedures are sound - if they generate a tree from a sequent which falsifies it, then the sequent was not a tautology.

2 Predicate Calculus

This involves the usual additions to the language: functions, constant symbols, variable which range over universes, and the $\forall$ and $\exists$ quantifiers.

An* interpretation *of a formula F in the first order logic consists of a non-empty domain D, and an assignment of "values" to each constant, function symbol, and predicate symbol occurring in F as follows:

1. To each constant we assign an element in D.
2. To each n-place function symbol, we assign a mapping from D^n to D. (Note that $D^n = \{(x_1, ..., x_n) \mid x_1 \in D, ..., x_n \in D\}$).
3. To each n-place predicate symbol, we assign a mapping from D^n to $\{\perp, \top\}$.

Sometimes, to emphasise the domain D, we speak of an interpretation of the formula *over* D. When we evaluate the truth value of a formula in an interpretation over the domain D, $\forall x$ will be interpreted as "for all elements x in D", and $\exists x$ as " there is an element in D".

For every interpretation of a formula over a domain D, the formula can be evaluated to $\perp$ or $\top$ according to the following rules:

1. If the truth values of L and R can be evaluated, then the truth value of $\neg R, L \wedge R, L \vee R, L \supset R, and L \equiv R$ can be evaluated as for propositional calculus.
2. $\forall x F$ evaluates to $\top$ if F evaluates to $\top$ for every d in D; otherwise it evaluates to $\perp$.
3. $\exists x F$ evaluates to $\top$ if F evaluates to $\top$ for at least one d in D; otherwise it evaluates to $\perp$.

Notice that this will not cope with free variables. It is common to demand that variables be quantified, and sometimes we *close* a formula by universally quantifying any free variables.

Example: Consider $\forall x P(x)$ and $\exists x \neg P(x)$ with the interpretation
Domain: $D = \{1, 2\}$
Assignment for P:

$P(1)$ $P(2)$
$\top$ $\bot$

Then using the rules, $\forall x P(x)$ is $\bot$ and $\exists x \neg P(x)$ is $\top$.

Termination can no longer be guaranteed of proofs, now that predicates take the place of propositions, and they are parameterised over universes which may be infinite, so there are an infinite number of symbols to be assigned a truth value in interpretations. Also, the inference rules (see below) do not reduce the number of connectives.

Some definitions:

- *A formula G is* **consistent(satisfiable)** *if and only if there exists an interpretation I such that G is evaluated to* $\top$ *in I. I a formula G is* $\top$ *in an interpretation I, we say that I is a* **model** *of G and I satisfies* G.
- *A formula G is* **inconsistent(unsatisfiable)** *if and only if there exists no interpratation that satisfies G.*
- *A formula G is* **valid** *if and only if every interpretation of G satisfies G.*
- *A formula G is a* **logical consequence** of formulas $F_1, ..., F_n$ if and only if for every interpretation I, if $F_1 \wedge ... \wedge F_n$ is true in I, G is also true in I.

We also need rules of inference for these new connectives. So continuing in the Gentzen Sequent Calculus we use:

$$\frac{\Gamma, A[t/x], \forall x A, \Delta \rightarrow \Lambda}{\Gamma, \forall x A, \Delta \rightarrow \Lambda} \qquad \frac{\Gamma \rightarrow \Delta, A[y/x], \Lambda}{\Gamma \rightarrow \Delta, \forall x A, \Lambda}$$

$$\frac{\Gamma, A[y/x], \Delta \rightarrow \Lambda}{\Gamma, \exists x A, \Delta \rightarrow \Lambda} \qquad \frac{\Gamma \rightarrow \Delta, A[t/x], \exists x A, \Lambda}{\Gamma \rightarrow \Delta, \exists x A, \Lambda}$$

N.B. in the rules y is a variable free for x in A, and not free in A unless $y = x$. t is any term free for x in A. A term t is **free for** x **in** A if either:

- A is atomic or
- A is $B \wedge C, B \vee C, B \supset C, \neg C$ or $B \equiv C$, and t is free for x in B and C or
- A is $\forall y B$ or $\exists y B$ and either $x = y$ or $x \neq y$, y is not a free variable in t and t is free for x in B.

Axioms still take the form of a sequent where the same formula appears on the left and right of $\rightarrow$.

Here is an example proof (some multiple applications of quantifier rules have been collapsed into one - where an instance is introduced, many could be introduced, and have been simultaneously):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \rightarrow P, \exists z Q(z) \qquad \dfrac{Q(y_1), P \rightarrow Q(y_0), Q(y_1), \exists z Q(z)}{Q(y_1), P \rightarrow \exists z Q(z)}}{P, P \supset Q(y_1) \rightarrow \exists z Q(z)}}{P \supset Q(y_1) \rightarrow P \supset \exists z Q(z)}}{\exists x (P \supset Q(x)) \rightarrow P \supset \exists z Q(z)}}{\rightarrow \exists x (P \supset Q(x)) \supset (P \supset \exists z Q(z))}$$

We can extend the search and expand algorithms we had before to generate such proofs (though there is no longer a guarantee of termination). But we now have the obligation of managing the variables and constants. Here are the algorithms as adapted for a system without function symbols.

In the following, $TERM_0$ is essentially a list of terms in the current sequent. Initially it is the constants and free variables, and as more free variables are introduced it is updated to include them. If there are no constants or free variables to start with, it is initialised to contain just y_0. If there are some constants or free variables, let them be $< u_0, ..., u_p >$ say. $AVAIL_0$ is an infinite list of free unused variables, say $< y_1, ..., y_n, ... >$.

This is really an exhaustive growing of the tree backwards, choosing what to introduce by knowing what's already around (in $TERM_0$), and introducing all of them if possible. If a new variable is introduced, that is noted and used for futher introductions later. It becomes important to note what constants and free variables have been used with which quantifiers.

Every time a ∀:right or a ∃:left rule is applied, as the variable y, the first element of $AVAIL_0$ is used. It is appended to $TERM_0$, and deleted from $AVAIL_0$.

When a ∀:left or a ∃:right is used, each of the terms in $TERM_0$ is used that have not been used with that formula before. So $TERM_0$ has to be a list of records, each of which is a constant or variable, and a list of the quantified formulae for which it has been used as t in a ∀:left or a ∃:right rule.

If a sequent has the property that all the formulae in it are either atomic or of the form $\forall xA$ or $\exists xA$ such that all the terms in $TERM_0$ have been used with a ∀:left or ∃:right rule, and if this sequent is not an axiom, then it will never become one, and it need no longer be expanded. If this happens, or if the sequent is an axiom, that leaf is closed.

During the search round, the same substitutions must be performed for all occurrences of a formula, so a local variable to the procedure is used, $TERM_1$, to store $TERM_0$ plus that round's updates. This replaces $TERM_0$ on the next round.

```
procedure search(Γ → Δ:sequent; var  T:tree) ;
    begin
        let T be the one-node tree labelled with Γ → Δ ;
        Let TERM0 be << u0, nil >, ..., < up, nil >>
        and let AVAIL0 be < y1, y2, ... > (as explained above)
        while not all leaves of T are finished do
            TERM1 :=TERM0 ; T0 := T ;
            for each leaf node of T0 do
                if not finished(node) then
                    expand(node, T)
                endif
            endfor;
            TERM0 := TERM1
        endwhile;
        if all leaves are closed
        then
```

```
            write( 'T is a proof of Γ → Δ' )
        else
            write( Γ → Δ is falsifiable' )
        endif
    end

procedure expand(node:tree-address ; variiT :tree );
    begin
        let A_1, ..., A_m → B_1, ..., B_n be the label of the node;
        let S be the one-node tree labelled with A_1, ..., A_m → B_1, ..., B_n;
        for i := 1 to m do
            if non-atomic(A_i) then
                it grow-left(A_i,S)
            endif
        endfor;
        for i := 1 to n do
            if non-atomic(B_i) then
                it grow-right(B_i,S)
            endif
        endfor;
        T:= dosubstitution(T,node,S)
    end

procedure grow-left(A:formula; varS:tree);
    begin
        case A of
            B ∧ C,B ∨ C
            B ⊃ C, ¬B:  extend every non-axiom leaf of S using the
                        left rule corresponding to the main
                        propositional connective;
            ∀xB:        for every term u_k with a record in TERM_0
                        such that A is not amongst u_k's list of formulae do
                            extend every non-axiom leaf of S by applying
                            the ∀: left rule using the term u_k
                            as one of the terms of the rule, and append A to u_k's list
                        endfor;
            existsxB:   extend every non-axiom leaf of S by applying
                        the ∃: left rule using the first element of AVAIL_0 for the new variable y;
                        append < y, nil > to TERM_1;
                        remove y from AVAIL_0;
        endcase
    end

procedure grow-right(A:formula; varS:tree);
    begin
```

case A **of**

$B \wedge C, B \vee C$

$B \supset C, \neg B$: *extend every non-axiom leaf of S using the right rule corresponding to the main propositional connective;*

$\exists x B$: **for** *every term* u_k *with a record in* $TERM_0$ *such that A is not amongst* u_k*'s list of formulae* **do**
extend every non-axiom leaf of S by applying the $\exists$*: right rule using the term* u_k *as one of the terms of the rule, and append A to* u_k*'s list*
endfor;

$forallxB$: *extend every non-axiom leaf of S by applying the* $\forall$*: right rule using the first element of* $AVAIL_0$ *for the new variable y;*
append $< y, nil >$ *to* $TERM_1$;
remove y from $AVAIL_0$;

endcase

end

Here's a proof it could produce:

$$\dfrac{\dfrac{P \rightarrow P, \forall z Q(z) \quad \dfrac{Q(y_1), P \rightarrow Q(y_2)}{Q(y_1), P \rightarrow \forall z Q(z)}}{P, P \supset Q(y_1) \rightarrow \forall z Q(z)}}{\dfrac{P \supset Q(y_1) \rightarrow P \supset \forall z Q(z)}{\dfrac{\exists x (P \supset Q(x)) \rightarrow P \supset \forall z Q(z)}{\rightarrow \exists x (P \supset Q(x)) \supset (P \supset \forall z Q(z))}}}$$

Without functions, this algorithm terminates for valid formulae, but otherwise may go on infinitely.

3 Herbrand's Theorem

In spite of various attempts to reduce the amount of search involved in finding proofs or counterexamples, the algorithms considered so far still leave us with the problem of validity defined over infinite universes and infinitely many interpretations. This is a major problem. Herbrand's theorem is a key to it. It says how to define a particular universe for a formula, and corresponding interpretations in which an unsatisfiable formula is guaranteed to be unsatisfiable if and only if it is actually unsatisfiable for all interpretations. This makes the task of establishing unsatisfiability a lot more manageable. It opens up considerably improved possiblities for automatic proof.

So if you're interested in proving some (closed) theorem *Thm* from some list of (closed) axioms *Ax*:

Thm is a logical consequence of *Ax*

iff

$Ax \wedge \neg Thm$ is unsatisfiable

iff

S, the clausal form of $Ax \wedge \neg Thm$ is unsatisfiable

iff

$\exists$ a contradiction consisting of a finite conjunction of instances of clauses of S.

(Herbrand's Theorem).

This theorem supports the soundness and completeness of resolution.

3.1 Unsatisfiablity

Certain kinds of normal form make unsatisfiablity easy to detect, as we've noted before. We're looking for something of the form $P \wedge \neg P$ or $Q(a) \wedge \neg Q(a)$ or a more elaborate variant say $P \wedge (\neg P \vee Q) \wedge \neg Q$. If we cast our formula to be refuted as a conjunction of clauses, then if part of this conjunction is unsatisfiable, the whole thing is. Each conjunct is called a **clause**. For unsatisfiablity, a clause must be false in *every* interpretation. But at least we can reduce the search to interpretations over 1 domain - the Herbrand Universe.

Before doing this however, we need to do a bit more work on the formula. We need to take all the quantifiers out to the front, conjunctive normal form it, and skolemize. This gets you a formula whose only variables are universally quantified. The resulting formula is unsatisfiable if and only if the original was. Though if the original formula was satisfiable the resulting formula is not in general equivalent to the original.

The advantage of this is that we can look at any of these (implicitly universally quantified) formulae as a super-conjunction of all the instances of it with elements of the universe or domain substituted for the variables. I.e. in

$$\forall x.P(x) \wedge (\neg P(x) \vee Q(f(x))) \wedge \neg Q(f(a))$$

which normal forms to:

$$P(x) \wedge (\neg P(x) \vee Q(f(x))) \wedge \neg Q(f(a))$$

the x is still (implicitly) universally quantified, and as x ranges over some universe, e.g. $e_1, e_2, e_3, \ldots$ we have to test the unsatisfiability of:

$$P(e_1) \wedge (\neg P(e_1) \vee Q(f(e_1))) \wedge \neg Q(f(a))$$

$$\wedge$$

$$P(e_2) \wedge (\neg P(e_2) \vee Q(f(e_2))) \wedge \neg Q(f(a))$$

$$\wedge$$

$$P(e_3) \wedge (\neg P(e_3) \vee Q(f(e_3))) \wedge \neg Q(f(a))$$

$$\wedge$$

etc.

But if we find some part of this to be unsatisfiable, the whole thing is, necessarily, and we need look no further.

3.2 Converting the Formula to Clausal Form

1. Replace all $\equiv$ using $\supset$ $\quad A \Leftrightarrow B \models (A \rightarrow B) \wedge (B \rightarrow A)$.
2. Replace all $\supset$ by using $F \supset G = \neg F \vee G$
3. Drive the $\neg$s down to the atoms:

$$\neg\neg F = F$$

$$\neg(F \vee G) = \neg F \wedge \neg G$$
$$\neg(F \wedge G) = \neg F \vee \neg G$$
$$\neg\forall x F(x) = \exists x \neg F(x)$$
$$\neg\exists x F(x) = \forall x \neg F(x)$$

4. Move all the quantifiers to the left:

$$(Qx F(x)) \vee G = Qx(F(x) \vee G)$$
$$(Qx F(x)) \wedge G = Qx(F(x) \wedge G)$$
$$\forall x F(x) \wedge \forall x H(x) = \forall x(F(x) \wedge H(x))$$
$$\exists x F(x) \vee \exists x H(x) = \exists x(F(x) \vee H(x))$$
$$Q_1 x F(x) \vee Q_2 x H(x) = Q_1 x Q_2 z(F(x) \vee H(z))$$
$$Q_3 x F(x) \wedge Q_4 x H(x) = Q_3 x Q_4 z(F(x) \wedge H(z))$$

   where x doesn't appear in a formula unless indicated, and Q stands for either quantifier.

5. Make the quantified formula conjunctive normal form, distributing *and* over *or*. The previous steps were necessary to make this possible.

6. Skolemize. This captures the information about universal and existential quantification and embeds it in the quantified formula. Each existentially quantified variable is replaced by a new function symbol with each of the preceding universally quantified variables as arguments, preserving the information about the dependency of existentially quantified variables on the universal quantification they lie in. If there are no universal quantifiers preceding an existential quantifier, its variable is replaced throughout by a nullary function, i.e. a new constant. The quantifier symbols are then removed. E.g.

$$\forall x \exists y \exists z(\neg P(x, y) \vee R(x, y, z)) \wedge (Q(x, z) \vee R(x, y, z))$$

   becomes

$$(\neg P(x, f_y(x)) \vee R(x, f_y(x), f_z(x))) \wedge (Q(x, f_z(x)) \vee R(x, f_y(x), f_z(x)))$$

Gilmore [1960] was one of the first persons to implement Herbrand's procedure on a computer. Since a formula is valid if and only if its negation is inconsistent, his program was designed to detect the inconsistency of the negation of the given formula. During the execution of his program, propositional formulas are generated that are periodically tested for inconsistency. If the negation of the given formula is inconsistent, his program will eventually detect this fact. Gilmore's program managed to prove a few simple formulas, but encountered decisive difficulties with most other formulas of the first order logic. Careful studies of his program revealed that his method of testing the inconsistency of a propositional formula is inefficient. Gilmore's method was improved by Davis and Putnam [1960] a few months after his result was published. However, their improvement was still not enough. Many valid formulas of the first-order logic still could not be proved by computers in a reasonable amount of time.

A major breakthrough was made by Robinson [1965a], who introduced the so-called resolution principle. Resolution proof procedure is much more efficient than any earlier procedure. Since the introduction of the resolution principle, several refinements have been suggested in attempts to further increase its efficiency. Some of these refinements are semantic resolution [Slagle, 1967; Meltzer, 1966; Robinson, 1965b; Kowalski and Hayes, 1969], lock resolution [Boyer, 1971], linear resolution [Loveland, 1970a, b; Luckham, 1970; Anderson and Bledsoe, 1970; Yates *et al.*, 1970; Reiter, 1971; Kowalski and Kuehner, 1970], unit resolution [Wos *et al.*, 1964; Chang, 1970a], and set-of-support strategy [Wos *et al.*, 1965]. In this chapter, we shall first prove Herbrand's theorem. The resolution principle and some refinements of the resolution principle will be discussed in subsequent chapters.

4.2 SKOLEM STANDARD FORMS

Herbrand's and the resolution proof procedures that are to be discussed later in this book are actually refutation procedures. That is, instead of proving a formula valid, they prove that the negation of the formula is inconsistent. This is just a matter of convenience. There is no loss of generality in using refutation procedures. Furthermore, these refutation procedures are applied to a "standard form" of a formula. This standard form was introduced by Davis and Putnam [1960], and will be used throughout this book.

Essentially what Davis and Putnam did was to exploit the following ideas:

1. A formula of the first-order logic can be transformed into prenex normal form where the matrix contains no quantifiers and the prefix is a sequence of quantifiers.

2. The matrix, since it does not contain quantifiers, can be transformed into a conjunctive normal form.
3. Without affecting the inconsistency property, the existential quantifiers in the prefix can be eliminated by using Skolem functions.

In Chapter 3, we have already discussed how to transform a formula into a prenex normal form. By the techniques given in Chapter 2, we also know how to transform the matrix into a conjunctive normal form. We now discuss how to eliminate the existential quantifiers.

Let a formula F be already in a prenex normal form $(Q_1x_1)\cdots(Q_nx_n)M$, where M is in a *conjunctive* normal form. Suppose Q_r is an existential quantifier in the prefix $(Q_1x_1)\cdots(Q_nx_n)$, $1 \leqslant r \leqslant n$. If no universal quantifier appears before Q_r, we choose a new constant c different from other constants occurring in M, replace all x_r appearing in M by c, and delete (Q_rx_r) from the prefix. If $Q_{s_1}, \ldots, Q_{s_m}$ are all the universal quantifiers appearing before Q_r, $1 \leqslant s_1 < s_2 \cdots < s_m < r$, we choose a new m-place function symbol f different from other function symbols, replace all x_r in M by $f(x_{s_1}, x_{s_2}, \ldots, x_{s_m})$, and delete (Q_rx_r) from the prefix. After the above process is applied to all the existential quantifiers in the prefix, the last formula we obtain is a *Skolem standard form* (*standard form* for short) of the formula F. The constants and functions used to replace the existential variables are called *Skolem* functions.

Example 4.1

Obtain a standard form of the formula

$$(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)\,P(x, y, z, u, v, w).$$

In the above formula, $(\exists x)$ is preceded by no universal quantifiers, $(\exists u)$ is preceded by $(\forall y)$ and $(\forall z)$, and $(\exists w)$ by $(\forall y)$, $(\forall z)$ and $(\forall v)$. Therefore, we replace the existential variable x by a constant a, u by a two-place function $f(y, z)$, and w by a three-place function $g(y, z, v)$. Thus, we obtain the following standard form of the formula:

$$(\forall y)(\forall z)(\forall v)\,P(a, y, z, f(y, z), v, g(y, z, v)).$$

Example 4.2

Obtain a standard form of the formula

$$(\forall x)(\exists y)(\exists z)((\sim P(x, y) \wedge Q(x, z)) \vee R(x, y, z)).$$

First, the matrix is transformed into a conjunctive normal form:

$$(\forall x)(\exists y)(\exists z)((\sim P(x, y) \vee R(x, y, z)) \wedge (Q(x, z) \vee R(x, y, z))).$$

Then, since $(\exists y)$ and $(\exists z)$ are both preceded by $(\forall x)$, the existential variables y and z are replaced, respectively, by one-place functions $f(x)$ and $g(x)$.

To show this does not necessarily produce a formula which is equivalent to the original, take the case $\exists x P(x)$. This normal forms to $P(a)$. If a domain were $1, 2$, and we consider the interpretation in which a is assigned to 1, and the assignments for P are $P(1) = \bot$ and $P(2) = \top$ then the original formula was true in the $1, 2$ interpretation, but the converted one is false in this interpretation.

The important thing is that unsatisfiability is preserved by this transformation.

3.3 Herbrand Universe

Although unsatisfiability is characterised in terms of interpretations, the actual contradiction tests we use are syntactic ($P \wedge \neg P$ etc). So, intuitively, you might feel this could be used somehow. That if anything is going to supply the required complementary literals, it's the symbols there in the formula already.

The Herbrand Universe is the set of all constants appearing in S, our clausally formed formula, and all terms which can be created from them using the function symbols in S. If there are no constants, create one, a say. For example here are some formulae and their Herbrand Universes:

$P(a) \wedge (\neg P(x) \vee P(f(x)))$	$a, f(a), f(f(a)), \ldots$
$(P(x) \vee Q(x)) \wedge R(z)$	a
$P(f(x), a, g(y), b)$	$a, b, f(a), f(b), g(a), g(b),$ $f(f(a)), f(f(b)), f(g(a)), f(g(b)),$ $g(f(a)), g(f(b)), g(g(a)), g(g(b))$, etc.

The **Herbrand Base** is the set of ground atoms formed from predicates occurring in S with arguments in the Herbrand Universe. A **ground instance** of a clause is the clause obtained by substituting members of the Herbrand Universe for all its variables.

So in the three examples above, the respective Herbrand Bases are:

$P(a) \wedge (\neg P(x) \vee P(f(x)))$	$P(a), P(f(a)), P(f(f(a))), \ldots$
$(P(x) \vee Q(x)) \wedge R(z)$	$P(a), Q(a), R(a)$
$P(f(x), a, g(y), b)$	$P(a,a,a,a), P(a,a,a,b), P(a,a,b,a), P(a,b,a,a),$ $P(b,a,a,a), P(a,a,b,b), P(a,b,b,a), P(b,b,a,a),$ $P(b,a,a,b), P(b,a,b,a), P(a,b,a,b), P(b,b,b,a),$ $P(b,b,a,b), P(b,a,b,b), P(a,b,b,b), P(b,b,b,b),$ $P(a,a,a,f(a)), \ldots$

A **Herbrand Interpretation** is as follows. For *S, a conjunction of clauses, H, its Herbrand Universe, and I, an interpretation of S over H* I is a Herbrand interpretation if:

- It maps constants in S to themselves
- For $h_1, \ldots, h_n \in H$. Assign any n-place function f an n-place function from H^n to H. E.g. if f is a function which takes two arguments, for any h_1, h_2 in H, define the result of f applied to them to be the term $f(h_1, h_2)$, which is, of course, in H. Notice that no evaluation is taking place here.

- There is no restriction on the assignment of truth values to predicate symbols. Any list of labels of truth values on the Herbrand Base will do.

We actually want the whole class of possible assignments of truth values to ground instances of predicates. So this defines a set of interpretations. These interpretations are enough for us to consider if we are only looking for unsatisfiability of formulae. That is the substance of Herbrand's theorem.

3.4 Constructing Herbrand Interpretations corresponding to particular interpretations

Suppose we have $S = P(x) \wedge Q(y, f(y, a))$
and an interpretation over **1,2**:

$a \mapsto 2$	$P(1) \mapsto \top$
	$P(2) \mapsto \bot$
$f(1,1) \mapsto 1$	$Q(1,1) \mapsto \bot$
$f(1,2) \mapsto 2$	$Q(1,2) \mapsto \top$
$f(2,1) \mapsto 2$	$Q(2,1) \mapsto \bot$
$f(2,2) \mapsto 1$	$Q(2,2) \mapsto \top$

Then we can define a *corresponding Herbrand Interpretation* over all the Herbrand Base by picking some element for the a of the Herbrand Universe to map to, and specifying that the Herbrand Interpretation echo this one:

Herbrand Base	Assigned to	Calculates to
$P(a)$	$P(2)$	$\bot$
$Q(a,a)$	$Q(2,2)$	$\top$
$P(f(a,a))$	$P(f(2,2))$	$P(1) = \top$
$Q(a,f(a,a))$	$Q(2,f(2,2))$	$Q(2,1) = \bot$
etc..		

In general, the procedure to construct Herbrand Interpretations *correponding* to (another) interpretation I:

1. Assign elements of the Herbrand Universe arbitrarily to some elements of I's domain. $h_i \mapsto d_i$.

2. Assign the truth value of $P(h_1, ..., h_n)$ where $h_i \in$ the Herbrand Universe, to be the truth value of $P(d_1, ..., d_n)$ in the interpretation for every P in the formula.

Then

1. If an interpretation I over a domain D satisfies a formula S, any corresponding Herbrand Interpretation will too.

2. A conjunction of clauses S is unsatisfiable if and only if S is false under all its Herbrand interpretations.

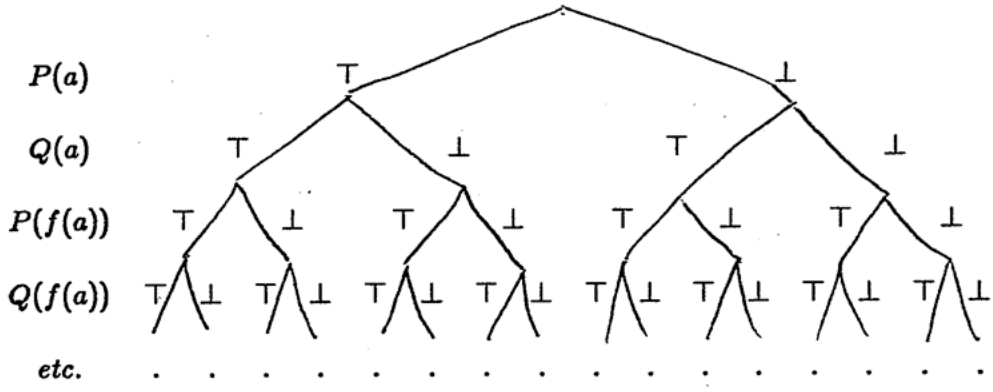
   - $\rightarrow$ is obvious

- ← Suppose S is false under all Herbrand interpretations, but not unsatisfiable. Then there is some interpretation I over D in which S is true. Construct its corresponding Herbrand Interpretation, S must be true in that too. But then we have a contradiction.

3.5 Establishing the Validity and Unsatisfiability of Formulae

So from the point of view of refutation proofs, where we're interested in unsatisfiability, we need only consider the Herbrand Interpretations. This is a considerable advance on having to consider all interpretations over all domains.

We can produce a semantic tree corresponding to the Herbrand Base, since it only contains ground instances of atomic formulae. In general this will of course be infinite, and the order in which it is generated by the methods described so far is not necessarily the most useful. Here is such a tree for $S = P(x) \wedge Q(f(x))$



A semantic tree is **complete** for S iff all its Herbrand Base elements are assigned in it.

Any node is a **partial interpretation.**

A node N is a **failure node** if its interpretation falsifies a ground instance of a clause of S, but none of its ancestors' interpretations falsify any ground instance of a clause of S.

The tree is **closed** iff every branch terminates with a failure node.

Example: $P(x) \wedge (\neg P(x) \vee Q(f(x))) \wedge \neg Q(f(a))$

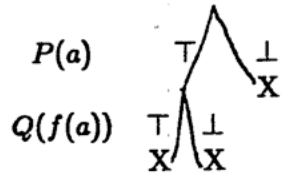
Remember this is:

$$P(e_1) \wedge (\neg P(e_1) \vee Q(f(e_1))) \wedge \neg Q(f(a))$$
$$\wedge$$
$$P(e_2) \wedge (\neg P(e_2) \vee Q(f(e_2))) \wedge \neg Q(f(a))$$
$$\wedge$$
$$P(e_3) \wedge (\neg P(e_3) \vee Q(f(e_3))) \wedge \neg Q(f(a))$$
$$\wedge$$

etc.

where $e_1, e_2, e_3, \ldots$ are all the elements of the Herbrand Universe. So if we reach a partial interpretation which makes any clause (e.g. $P(x)$ or $(Q(f(a)))$ or $\neg P(x) \vee Q(f(x))$) false, there's no point in expanding that branch further, since the whole formula is false for that interpretation whatever the rest of the truth assignments are.

Here is an incomplete semantic tree, but all of its leaves are failure nodes, so it is closed (and finite):



The branch corresponding to $P(a)$ false needs no further developement, because any extended interpretation would still include $P(a)$ being false, and the conjunction of ground instances is falsified by that.

3.6 Two Versions of Herbrand's Theorem

In the following, S is a formula in conjunctive normal form as described above.

1. *S is unsatisfiable iff corresponding to every complete semantic tree there is a finite closed semantic tree*

   - *unsatisfiable* $\rightarrow$ *closed*
     Let B be a branch in the tree and I_B its interpretation.
     I_B must falsify a ground instance of a clause C of S.
     C is finite, so there is a node N on B such that N's partial interpretation is enough. Prune the tree below N.
     Every branch can be pruned like this, so the tree must be closed.
   - *closed* $\rightarrow$ *unsatisfiable*
     Every branch must contain a failure node.
     So every interpretation falsifies S.

2. *S is unsatisfiable iff there is a finite unsatisfiable conjunction of ground instances of its clauses*

   - *unsatisfiable* $\rightarrow \exists$ *a finite unsatisfiable conjunction of ground instances of its clauses*
     Let T be S's semantic tree.
     Let T' be a corresponding finite closed semantic tree, which the first version of the theorem guarantees will exist.

Let S' be the conjunction of all the ground instances of clauses falsified at failure nodes of T'

S' is finite, and false in all interpretations, i.e. unsatisfiable.

- $\exists$ *a finite unsatisfiable conjunction of ground instances of S's clauses* $\rightarrow$ *unsatisfiable*

  Let S' be a finite unsatisfiable conjunction of ground clauses of S.

  Let I be any interpretation of S.

  Then I contains (or equals) an interpretation of S'

  But S' is unsatisfiable, so the contained interpretation must falsify it, and therefore so must I.

  But then I must also falsify S

The second of these is more commonly used.

This theorem is the theoretical basis for using the resolution rule of inference.

3.7 Using Herbrand's Theorem

The simple way to use the theorem (this is not resolution) is to generate the Herbrand universe progressively, and to create the corresponding increasing conjunction of ground clauses too, and check at each stage for unsatisfiability. If it is detected, stop, otherwise keep going, some larger conjunction may turn out unsatisfiable.

This is where the second version of the theorem turns out to be more useful. You can detect contradictions more easily than exploring whole semantic trees.

The progression of universes can be defined recursively. You have a set of constant symbols, and a set of function symbols from the formula. The first subset of the Herbrand Universe, H_0 is just the constant symbols. To create the next, apply all the function symbols to all the elements of H_0 in all possible ways, and take the union of the result and H_0. Do the same to create each subsequent set, apply the functions to all the elements of the previous set in all possible ways, and take the union of the result with the previous set.

E.g.

$$P(x) \wedge \neg P(f(f(a)))$$

H_0	a	$P(a) \wedge \neg P(f(f(a)))$
H_1	$a, f(a)$	$(P(a) \wedge \neg P(f(f(a)))) \wedge (P(f(a)) \wedge \neg P(f(f(a))))$
H_2	$a, f(a), f(f(a))$	$(P(a) \wedge \neg P(f(f(a)))) \wedge (P(f(a)) \wedge \neg P(f(f(a)))) \wedge (P(f(f(a))) \wedge \neg P(f(f(a))))$

At each stage the formula is essentially propositional, and therefore its (un)satisfiability is decidable. The semantic tree is finite for each conjunction of ground literals.

This is not very efficient though, with quite a small number of constants and functions, the size of the H_i can grow very rapidly. The semantic tree grows even more rapidly.

Various people looked at ways of making this more efficient.

3.8 Gilmore's Procedure

Take the conjunction of ground clauses at each stage of developement, turn it into disjunctive normal form, and there's a contradiction in all the disjuncts. If, not go on to the next, bigger conjunction.

3.9 Davis and Putnam's Procedure

1. Again, create a conjunction of ground instances of the formula, F.

   (a) Delete all clauses that are tautologies, i.e. they contain Q and $\neg Q$ for some Q.

   (b) If any of the clauses in F is a unit ground clause, P, i.e. P is a ground literal[1] then:

      i. If all of F's clause contain P, F is satisfiable, stop, try another conjunction of clauses.

      ii. If $\neg P$ is a unit clause, it is replaced by $\Box$, so, F is unsatisfiable, stop.

      iii. Otherwise, delete all the clauses which contain P, and all occurrences of $\neg P$ from the rest of the clauses.

   (c) There is a literal L in F, and no occurrence of $\neg L$, delete all clauses containing L.

   (d) If F can be put into the form

   $$(A_1 \vee L) \wedge ... \wedge (A_m \vee L) \wedge (B_1 \vee \neg L) \wedge ... \wedge (B_n \vee \neg L) \wedge R$$

   where A_i and B_i and R are free of L and $\neg L$, then for

   $$F_1 = A_1 \wedge ... \wedge A_m \wedge R$$

   and

   $$F_2 = B_1 \wedge ... \wedge B_m \wedge R$$

   F is unsatisfiable if and only if both F_1 and F_2 are, i.e. $F_1 \vee F_2$ is.

   (e) Repeat these procedures as long as possible.

2. If you haven't proved unsatisfiability, use the next H_i to create another conjunction, and repeat.

Examples:

$(P \vee Q \vee \neg R) \wedge (P \vee \neg Q) \wedge \neg P \wedge R \wedge U$
$(Q \vee \neg R) \wedge (\neg Q) \wedge R \wedge U$ rule (b) on $\neg P$
$\neg R \wedge R \wedge U$ rule (b) on $\neg Q$
$\Box \wedge U$ rule (b) on $\neg Q$
which is unsatisfiable, since it contains the empty clause (the OR of no literals).

[1] an atom or the negation of an atom

$(P \vee Q) \wedge \neg Q \wedge (\neg P \vee Q \vee \neg R)$
$P \wedge (\neg P \vee \neg R)$ rule (b) on $\neg Q$
$\neg R$ rule (b) on P
which is satisfiable, since using rule (b) on $\neg R$, we get the AND of no literals.

$(P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R)$
$(\neg Q \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R))$
$\vee (Q \wedge (Q \vee \neg R) \wedge (\neg Q \vee \neg R))$ rule (d) on P
$\neg R \vee \neg R$ rule (b) on Q and $\neg Q$
which is satisfiable by using rule (b) on $\neg R$

$(P \vee Q) \wedge (P \vee \neg Q) \wedge (R \vee Q) \wedge (R \vee \neg Q)$
$(R \vee Q) \wedge (R \vee \neg Q)$ rule (c) on P
which is satisfiable using rule (c) on R

3.10 More Efficient Procedures

The previous two methods still leave us with the overhead of generating the universe and creating lots of ground instances, many of which will not be needed. What would be preferable would be to detect inherent contradictions with minimal instance creation.

We want to detect potential contradictions. Think about:

$$P(x) \wedge \neg P(f(f(f(f(f(x))))))$$

We don't really want to generate the whole of:

$P(a)$ $\wedge$ $\neg P(f(f(f(f(f(a))))))$
$\wedge$
$P(f(a))$ $\wedge$ $\neg P(f(f(f(f(f(f(a)))))))$
$\wedge$
$P(f(f(a)))$ $\wedge$ $\neg P(f(f(f(f(f(f(f(a))))))))$
$\wedge$
$P(f(f(f(a))))$ $\wedge$ $\neg P(f(f(f(f(f(f(f(f(a)))))))))$
$\wedge$
$P(f(f(f(f(a)))))$ $\wedge$ $\neg P(f(f(f(f(f(f(f(f(f(a))))))))))$
$\wedge$
$P(f(f(f(f(f(a))))))$ $\wedge$ $\neg P(f(f(f(f(f(f(f(f(f(f(a)))))))))))$

We want to detect the contradiction yielded by the top right and bottom left literals above. They are complementary, one is the negation of the other. If we hadn't been doggedly generating Herbrand universe from the bottom, we could have foreseen this.

If you think about it, the identical naming of the x's in the original formula was spurious. The first one had to cover all possible values, so did the second, The two literals were ANDed together, and the implicit universal quantification ANDed all the instances. We can therefore standardise the variable names apart in separate clauses,

Thus, we obtain the following standard form of the formula:

$$(\forall x)\big((\sim P(x, f(x)) \vee R(x, f(x), g(x))) \wedge (Q(x, g(x)) \vee R(x, f(x), g(x)))\big).$$

Definition A *clause* is a finite disjunction of zero or more literals.

When it is convenient, we shall regard a set of literals as synonymous with a clause. For example, $P \vee Q \vee \sim R = \{P, Q, \sim R\}$. A clause consisting of r literals is called an *r-literal* clause. A one-literal clause is called a *unit* clause. When a clause contains no literal, we call it the *empty* clause. Since the empty clause has no literal that can be satisfied by an interpretation, the empty clause is always false. We customarily denote the empty clause by $\square$.

The disjunctions $\sim P(x,f(x)) \vee R(x,f(x),g(x))$ and $Q(x,g(x)) \vee R(x,f(x),g(x))$ in the standard form in Example 4.2 are clauses. *A set S of clauses is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.* By this convention, a standard form can be simply represented by a set of clauses. For example, the standard form of Example 4.2 can be represented by the set

$$\{\sim P(x,f(x)) \vee R(x,f(x),g(x)), Q(x,g(x)) \vee R(x,f(x),g(x))\}.$$

As we said in the beginning of this section, we can eliminate existential quantifiers without affecting the inconsistency property. This is shown in the following theorem.

Theorem 4.1 Let S be a set of clauses that represents a standard form of a formula F. Then F is inconsistent if and only if S is inconsistent.

Proof Without loss of generality, we may assume that F is in a prenex normal form, that is, $F = (Q_1x_1)\cdots(Q_nx_n)M[x_1, \ldots, x_n]$. (We use $M[x_1, \ldots, x_n]$ to indicate that the matrix M contains variables $x_1, \ldots, x_n$.) Let Q_r be the first existential quantifier. Let $F_1 = (\forall x_1)\cdots(\forall x_{r-1})(Q_{r+1}x_{r+1})\cdots(Q_nx_n)M[x_1, \ldots, x_{r-1}, f(x_1, \ldots, x_{r-1}), x_{r+1}, \ldots, x_n]$, where f is a Skolem function corresponding to x_r, $1 \leqslant r \leqslant n$. We want to show that F is inconsistent if and only if F_1 is inconsistent. Suppose F is inconsistent. If F_1 is consistent, then there is an interpretation I such that F_1 is true in I. That is, for all $x_1, \ldots, x_{r-1}$, there exists at least one element, which is $f(x_1, \ldots, x_{r-1})$, such that $(Q_{r+1}x_{r+1})\cdots(Q_nx_n)M[x_1, \ldots, x_{r-1}, f(x_1, \ldots, x_{r-1}), x_{r+1}, \ldots, x_n]$ is true in I. Thus, F is true in I, which contradicts the assumption that F is inconsistent. Therefore F_1 must be inconsistent. On the other hand, suppose that F_1 is inconsistent. If F is consistent, then there is an interpretation I over a domain D such that F is true in I. That is, for all $x_1, \ldots, x_{r-1}$, there exists an element x_r such that $(Q_{r+1}x_{r+1})\cdots(Q_nx_n)M[x_1, \ldots, x_{r-1}, x_r, x_{r+1}, \ldots, x_n]$ is true in I. Extend the interpretation I to include a function f that maps $(x_1, \ldots, x_{r-1})$ to x_r for all $x_1, \ldots, x_{r-1}$ in D, that is, $f(x_1, \ldots, x_{r-1}) = x_r$. Let

this extension of I be denoted by I'. Then, clearly, for all $x_1, \ldots, x_{r-1}$, $(Q_{r+1}x_{r+1})\cdots(Q_nx_n)M[x_1, \ldots, x_{r-1}, f(x_1, \ldots, x_{r-1}), x_{r+1}, \ldots, x_n]$ is true in I'. That is, F_1 is true in I', which contradicts the assumption that F_1 is inconsistent. Therefore F must be inconsistent. Assume there are m existential quantifiers in F. Let $F_0 = F$. Let F_k be obtained from F_{k-1} by replacing the first existential quantifier in F_{k-1} by a Skolem function, $k = 1, \ldots, m$. Clearly, $S = F_m$. Using the same arguments given above, we can show that F_{k-1} is inconsistent if and only if F_k is inconsistent for $k = 1, \ldots, m$. Therefore, we conclude that F is inconsistent if and only if S is inconsistent. Q.E.D.

Let S be a standard form of a formula F. If F is inconsistent, then by Theorem 4.1, $F = S$. If F is not inconsistent, we note that, in general, F is not equivalent to S. For example, let $F \triangleq (\exists x)P(x)$ and $S \triangleq P(a)$. Clearly, S is a standard form of F. However, let I be the interpretation defined below:

Domain: $D = \{1, 2\}$.

Assignment for a:

a
1

Assignment for P:

$P(1)$	$P(2)$
F	T

Then, clearly, F is true in I, but S is false in I. Therefore $F \neq S$.

It is noted that a formula may have more than one standard form. For the sake of simplicity, when we transform a formula F into a standard form S, we should replace existential quantifiers by Skolem functions that are as simple as possible. That is, we should use Skolem functions with the least number of arguments. This means that we should move existential quantifiers to the left as far as possible. Furthermore, if we have $F = F_1 \wedge \cdots \wedge F_n$, we can separately obtain a set S_i of clauses, where each S_i represents a standard form of F_i, $i = 1, \ldots, n$. Then, let $S = S_1 \cup \cdots \cup S_n$. By arguments similar to those given in the proof of Theorem 4.1, it is not difficult to see that F is inconsistent if and only if S is inconsistent.

Example 4.3

In this example, we shall show how to express the following theorem in a standard form:

If $x \cdot x = e$ for all x in group G, where $\cdot$ is a binary operator and e is the identity in G, then G is commutative.

We shall first symbolize the above theorem together with some basic axioms in group theory and then represent the negation of the above theorem by a set of clauses.

We know that group G satisfies the following four axioms:

A_1: $x, y \in G$ implies that $x \cdot y \in G$ (closure property);

A_2: $x, y, z \in G$ implies that $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ (associativity property);

A_3: $x \cdot e = e \cdot x = x$ for all $x \in G$ (identity property);

A_4: for every $x \in G$ there exists an element $x^{-1} \in G$ such that $x \cdot x^{-1} = x^{-1} \cdot x = e$ (inverse property).

Let $P(x, y, z)$ stand for $x \cdot y = z$ and $i(x)$ for x^{-1}. Then the above axioms can be represented by

A_1': $(\forall x)(\forall y)(\exists z) P(x, y, z)$

A_2': $(\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)(P(x, y, u) \wedge P(y, z, v) \wedge P(u, z, w) \rightarrow P(x, v, w))$
$\wedge (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)(P(x, y, u) \wedge P(y, z, v) \wedge P(x, v, w) \rightarrow P(u, z, w))$

A_3': $(\forall x) P(x, e, x) \wedge (\forall x) P(e, x, x)$

A_4': $(\forall x) P(x, i(x), e) \wedge (\forall x) P(i(x), x, e)$.

The conclusion of the theorem is

B: If $x \cdot x = e$ for all $x \in G$, then G is commutative, i.e., $u \cdot v = v \cdot u$ for all $u, v \in G$.

B can be represented by

B': $(\forall x) P(x, x, e) \rightarrow ((\forall u)(\forall v)(\forall w)(P(u, v, w) \rightarrow P(v, u, w)))$.

Now, the entire theorem is represented by the formula $F = A_1' \wedge \cdots \wedge A_4' \rightarrow B'$. Thus, $\sim F = A_1' \wedge A_2' \wedge A_3' \wedge A_4' \wedge \sim B'$. To obtain a set S of clauses for $\sim F$, we first obtain a set S_i of clauses for each axiom A_i', $i = 1, 2, 3, 4$ as follows.

S_1: $\{P(x, y, f(x, y))\}$

S_2: $\{\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(u, z, w) \vee P(x, v, w),$
$\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(x, v, w) \vee P(u, z, w)\}$

S_3: $\{P(x, e, x), P(e, x, x)\}$.

S_4': $\{P(x, i(x), e), P(i(x), x, e)\}$.

Since

$$\begin{aligned}
\sim B' &= \sim((\forall x) P(x, x, e) \rightarrow ((\forall u)(\forall v)(\forall w)(P(u, v, w) \rightarrow P(v, u, w))))\\
&= \sim(\sim(\forall x) P(x, x, e) \vee ((\forall u)(\forall v)(\forall w)(\sim P(u, v, w) \vee P(v, u, w))))\\
&= (\forall x) P(x, x, e) \wedge \sim((\forall u)(\forall v)(\forall w)(\sim P(u, v, w) \vee P(v, u, w)))\\
&= (\forall x) P(x, x, e) \wedge (\exists u)(\exists v)(\exists w)(P(u, v, w) \wedge \sim P(v, u, w)),
\end{aligned}$$

a set of clauses for $\sim B'$ is given below.

T: $\{P(x, x, e),$
$P(a, b, c),$
$\sim P(b, a, c)\}$.

Thus, the set $S = S_1 \cup S_2 \cup S_3 \cup S_4 \cup T$ is the set consisting of the following clauses:

(1) $P(x, y, f(x, y))$

(2) $\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(u, z, w) \vee P(x, v, w)$

(3) $\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(x, v, w) \vee P(u, z, w)$

(4) $P(x, e, x)$

(5) $P(e, x, x)$

(6) $P(x, i(x), e)$

(7) $P(i(x), x, e)$

(8) $P(x, x, e)$

(9) $P(a, b, c)$

(10) $\sim P(b, a, c)$.

In Example 4.3, we have shown how to obtain a set S of clauses for the formula $\sim F$. By Theorems 2.2 and 4.1, we know that F is valid if and only if S is inconsistent. As we said at the beginning of this section, we shall use refutation procedures to prove theorems. Thus, from here on, we shall assume that the input to a refutation procedure is always a set of clauses, such as the set S obtained in the above example. Furthermore, we shall use "unsatisfiable" ("satisfiable"), instead of "inconsistent" ("consistent"), for sets of clauses.

4.3 THE HERBRAND UNIVERSE OF A SET OF CLAUSES

By definition, a set S of clauses is unsatisfiable if and only if it is false under all interpretations over all domains. Since it is inconvenient and impossible

to consider all interpretations over all domains, it would be nice if we could fix on one special domain H such that S is unsatisfiable if and only if S is false under all the interpretations over this domain. Fortunately, there does exist such a domain, which is called the *Herbrand universe* of S, defined as follows.

Definition Let H_0 be the set of constants appearing in S. If no constant appears in S, then H_0 is to consist of a single constant, say $H_0 = \{a\}$. For $i = 0, 1, 2, \ldots$, let H_{i+1} be the union of H_i and the set of all terms of the form $f^n(t_1, \ldots, t_n)$ for all n-place functions f^n occurring in S, where t_j, $j = 1, \ldots, n$, are members of the set H_i. Then each H_i is called the *i-level constant set* of S, and H_∞, or $\lim_{i\to\infty} H_i$, is called the *Herbrand universe* of S.

Example 4.4

Let $S = \{P(a), \sim P(x) \vee P(f(x))\}$. Then

$$H_0 = \{a\}$$
$$H_1 = \{a, f(a)\}$$
$$H_2 = \{a, f(a), f(f(a))\}$$
$$\vdots$$
$$H_\infty = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}.$$

Example 4.5

Let $S = \{P(x) \vee Q(x), R(z), T(y) \vee \sim W(y)\}$. Since there is no constant in S, we let $H_0 = \{a\}$. There is no function symbol in S, hence $H = H_0 = H_1 = \cdots = \{a\}$.

Example 4.6

Let $S = \{P(f(x), a, g(y), b)\}$. Then

$$H_0 = \{a, b\}$$
$$H_1 = \{a, b, f(a), f(b), g(a), g(b)\}$$
$$H_2 = \{a, b, f(a), f(b), g(a), g(b), f(f(a)), f(f(b)), f(g(a)), f(g(b)), g(f(a)), g(f(b)), g(g(a)), g(g(b))\}$$
$$\vdots$$

In the sequel, by expression we mean a term, a set of terms, an atom, a set of atoms, a literal, a clause, or a set of clauses. When no variable appears in an expression, we sometimes call the expression a *ground expression* to emphasize this fact. Thus we may use a ground term, a ground atom, a ground literal, and a ground clause to mean that no variable occurs

in the respective expressions. Furthermore, a *subexpression* of an expression E is an expression that occurs in E.

Definition Let S be a set of clauses. The set of ground atoms of the form $P^n(t_1, \ldots, t_n)$ for all n-place predicates P^n occurring in S, where $t_1, \ldots, t_n$ are elements of the Herbrand universe of S, is called the *atom set*, or the *Herbrand base* of S.

Definition A *ground instance* of a clause C of a set S of clauses is a clause obtained by replacing variables in C by members of the Herbrand universe of S.

Example 4.7

Let $S = \{P(x), Q(f(y)) \vee R(y)\}$. $C = P(x)$ is a clause in S and $H = \{a, f(a), f(f(a)), \ldots\}$ is the Herbrand universe of S. Then $P(a)$ and $P(f(f(a)))$ are both ground instances of C.

We now consider interpretations over the Herbrand universe. Let S be a set of clauses. As discussed in Chapter 3, an interpretation over the Herbrand universe of S is an assignment of constants, function symbols, and predicate symbols occurring in S. In the following, we shall define a special interpretation over the Herbrand universe of S, called the H-interpretation of S.

Definition Let S be a set of clauses; H, the Herbrand universe of S; and I, an interpretation of S over H. I is said to be an *H-interpretation* of S if it satisfies the following conditions:

1. I maps all constants in S to themselves.
2. Let f be an n-place function symbol and $h_1, \ldots, h_n$ be elements of H. In I, f is assigned a function that maps $(h_1, \ldots, h_n)$ (an element in H^n) to $f(h_1, \ldots, h_n)$ (an element in H).

There is no restriction on the assignment to each n-place predicate symbol in S. Let $A = \{A_1, A_2, \ldots, A_n, \ldots\}$ be the atom set of S. An H-interpretation I can be conveniently represented by a set

$$I = \{m_1, m_2, \ldots, m_n, \ldots\}$$

in which m_j is either A_j or $\sim A_j$ for $j = 1, 2, \ldots$. The meaning of this set is that if m_j is A_j, then A_j is assigned "true"; otherwise, A_j is assigned "false."

Example 4.8

Consider the set $S = \{P(x) \vee Q(x), R(f(y))\}$. The Herbrand universe H of S is $H = \{a, f(a), f(f(a)), \ldots\}$. There are three predicate symbols: P, Q, and R. Hence the atom set of S is

$$A = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)), \ldots\}.$$

Some H-interpretations for S are as follows:

$$I_1 = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)), \ldots\}$$

$$I_2 = \{\sim P(a), \sim Q(a), \sim R(a), \sim P(f(a)), \sim Q(f(a)), \sim R(f(a)), \ldots\}$$

$$I_3 = \{P(a), Q(a), \sim R(a), P(f(a)), Q(f(a)), \sim R(f(a)), \ldots\}.$$

An interpretation of a set S of clauses does not necessarily have to be defined over the Herbrand universe of S. Thus an interpretation may not be an H-interpretation. For example, let $S = \{P(x), Q(y, f(y, a))\}$. If the domain is $D = \{1, 2\}$, then the following is an interpretation of S.

$D = \{1, 2\}$.

a	$f(1,1)$	$f(1,2)$	$f(2,1)$	$f(2,2)$
2	1	2	2	1

$P(1)$	$P(2)$	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
T	F	F	T	F	T

For an interpretation such as the one defined above, we can define an H-interpretation I^* corresponding to I. We use the above example to illustrate this point. First, we find the atom set of S,

$$A = \{P(a), Q(a, a), P(f(a, a)), Q(a, f(a, a)), Q(f(a, a), a), Q(f(a, a), f(a, a)), \ldots\}.$$

Next, we evaluate each member of A by using the above table.

$$P(a) = P(2) = F$$

$$Q(a, a) = Q(2, 2) = T$$

$$P(f(a, a)) = P(f(2, 2)) = P(1) = T$$

$$Q(a, f(a, a)) = Q(2, f(2, 2)) = Q(2, 1) = F$$

$$Q(f(a, a), a) = Q(f(2, 2), 2) = Q(1, 2) = T$$

$$Q(f(a, a), f(a, a)) = Q(f(2, 2), f(2, 2)) = Q(1, 1) = F$$

$$\vdots$$

Therefore, the H-interpretation I^* corresponding to I is

$$I^* = \{\sim P(a), Q(a, a), P(f(a, a)), \sim Q(a, f(a, a)), Q(f(a, a), a), \sim Q(f(a, a), f(a, a)), \ldots\}.$$

In case there is no constant in S, the element a that is used to initiate the Herbrand universe of S can be mapped into any element of the domain

D. In this case, if there is more than one element in D, then there is more than one H-interpretation corresponding to I. For example, let $S = \{P(x), Q(y, f(y, z))\}$ and let an interpretation I for S be as follows:

$D = \{1, 2\}$

$f(1,1)$	$f(1,2)$	$f(2,1)$	$f(2,2)$
1	2	2	1

$P(1)$	$P(2)$	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
T	F	F	T	F	T

Then the two H-interpretations corresponding to I are

$$I^* = \{\sim P(a), Q(a,a), P(f(a,a)), \sim Q(a, f(a,a)), Q(f(a,a),a), \sim Q(f(a,a),f(a,a)), \ldots\} \quad \text{if } a = 2,$$

$$I^* = \{P(a), \sim Q(a,a), P(f(a,a)), \sim Q(a,f(a,a)), \sim Q(f(a,a),a), \sim Q(f(a,a),f(a,a)), \ldots\} \quad \text{if } a = 1.$$

We can formalize the concepts mentioned above as follows:

Definition Given an interpretation I over a domain D, an *H-interpretation I^* corresponding to I* is an H-interpretation that satisfies the following condition:

Let $h_1, \ldots, h_n$ be elements of H (the Herbrand universe of S). Let every h_i be mapped to some d_i in D. If $P(d_1, \ldots, d_n)$ is assigned $T(F)$ by I, then $P(h_1, \ldots, h_n)$ is also assigned $T(F)$ in I^*.

In fact, it is not hard to prove the following lemma. The proof is left as an exercise.

Lemma 4.1 If an interpretation I over some domain D satisfies a set S of clauses, then any one of the H-interpretations I^* corresponding to I also satisfies S.

Theorem 4.2 A set S of clauses is unsatisfiable if and only if S is false under all the H-interpretations of S.

Proof ($\Rightarrow$) The first half of the above theorem is obvious since, by definition, S is unsatisfiable if and only if S is false under all the interpretations over *any* domain.

($\Leftarrow$) To prove the second half of the above theorem, assume that S is false under all the H-interpretations of S. Suppose S is not unsatisfiable. Then there is an interpretation I over some domain D such that S is true

under I. Let I^* be an H-interpretation corresponding to I. According to Lemma 4.1, S is true under I^*. This contradicts the assumption that S is false under all the H-interpretations of S. Therefore, S must be unsatisfiable. Q.E.D.

Thus we have obtained the objective stated at the beginning of this section. That is, we need consider only interpretations over the Herbrand universe, or more strongly, H-interpretations, for checking whether or not a set of clauses is unsatisfiable. Because of the above theorem, from here on, whenever we mention an interpretation, we mean an H-interpretation.

Let $\varnothing$ denote the empty set. Each of the following observations is obvious. We shall leave their proofs to the reader.

1. A ground instance C' of a clause C is satisfied by an interpretation I if and only if there is a ground literal L' in C' such that L' is also in I, that is, $C' \cap I \neq \varnothing$.
2. A clause C is satisfied by an interpretation I if and only if every ground instance of C is satisfied by I.
3. A clause C is falsified by an interpretation I if and only if there is at least one ground instance C' of C such that C' is not satisfied by I.
4. A set S of clauses is unsatisfiable if and only if for every interpretation I, there is at least one ground instance C' of some clause C in S such that C' is not satisfied by I.

Example 4.9

1. Consider the clause $C = \sim P(x) \vee Q(f(x))$. Let I_1, I_2, and I_3 be defined as follows:

$$I_1 = \{\sim P(a), \sim Q(a), \sim P(f(a)), \sim Q(f(a)), \sim P(f(f(a))), \sim Q(f(f(a))), \ldots\}$$

$$I_2 = \{\ P(a),\ Q(a),\ P(f(a)),\ Q(f(a)),\ P(f(f(a))),\ Q(f(f(a))), \ldots\}$$

$$I_3 = \{\ P(a), \sim Q(a),\ P(f(a)), \sim Q(f(a)),\ P(f(f(a))), \sim Q(f(f(a))), \ldots\}.$$

The reader should be able to see that C is satisfied by I_1 and I_2, but falsified by I_3.

2. Consider $S = \{P(x), \sim P(a)\}$. There are only two H-interpretations:

$$I_1 = \{P(a)\} \quad \text{and} \quad I_2 = \{\sim P(a)\}.$$

S is falsified by both H-interpretations, and therefore is unsatisfiable.

4.4 SEMANTIC TREES

Having introduced the Herbrand universe, we now consider semantic trees [Robinson, 1968a; Kowalski and Hayes, 1969]. It will be seen in the sequel that finding a proof for a set of clauses is equivalent to generating a semantic tree.

Definition If A is an atom, then the two literals A and $\sim A$ are said to be each other's *complement*, and the set $\{A, \sim A\}$ is called a *complementary pair.*

We note that a clause is a tautology if it contains a complementary pair. In the sequel, when we use "tautology," we shall specifically mean a clause that is a tautology.

Definition Given a set S of clauses, let A be the atom set of S. A *semantic tree* for S is a (downward) tree T, where each link is attached with a finite set of atoms or negations of atoms from A in such a way that:

i. For each node N, there are only finitely many immediate links $L_1, \ldots, L_n$ from N. Let Q_i be the conjunction of all the literals in the set attached to L_i, $i = 1, \ldots, n$. Then $Q_1 \vee Q_2 \vee \cdots \vee Q_n$ is a valid propositional formula.

ii. For each node N, let $I(N)$ be the union of all the sets attached to the links of the branch of T down to and including N. Then $I(N)$ does not contain any complementary pair.

Definition Let $A = \{A_1, A_2, \ldots, A_k, \ldots\}$ be the atom set of a set S of clauses. A semantic tree for S is said to be *complete* if and only if for every tip node N of the semantic tree, that is, a node that has no links sprouting from it, $I(N)$ contains either A_i or $\sim A_i$ for $i = 1, 2, \ldots$.

Example 4.10

Let $A = \{P, Q, R\}$ be the atom set of a set S of clauses. Then each one of the two trees in Fig. 4.1 is a complete semantic tree for S. (See p. 58.)

Example 4.11

Consider $S = \{P(x), P(a)\}$. The atom set of S is $\{P(a)\}$. A complete semantic tree for S is shown in Fig. 4.2. (See p. 58.)

Example 4.12

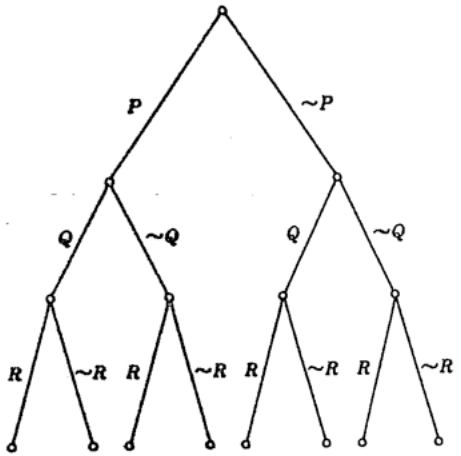
Consider $S = \{P(x), Q(f(x))\}$. The atom set of S is

$$\{P(a), Q(a), P(f(a)), Q(f(a)), P(f(f(a))), Q(f(f(a))), \ldots\}.$$

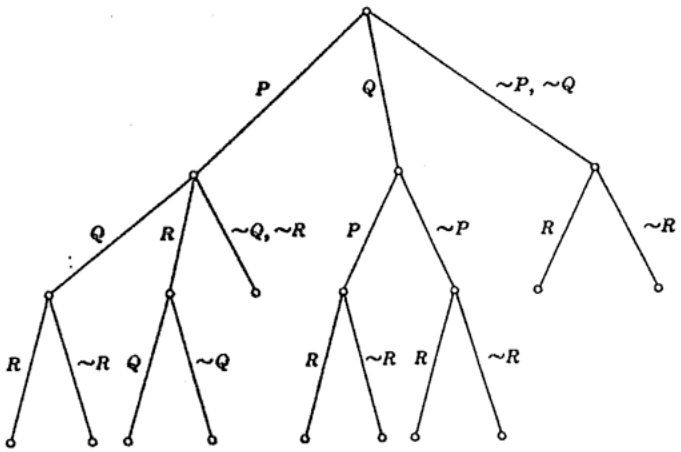
Fig. 4.3 shows a semantic tree for S.

It is noted that for each node N in a semantic tree for S, $I(N)$ is a subset of some interpretation for S. For this reason, $I(N)$ will be called a *partial interpretation* for S.

When the atom set A of a set S of clauses is infinite, any complete semantic tree for S will be infinite. As is easily seen, a complete semantic



(a)



(b)

Figure 4.1

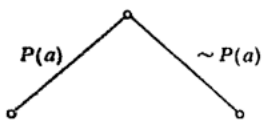


Figure 4.2

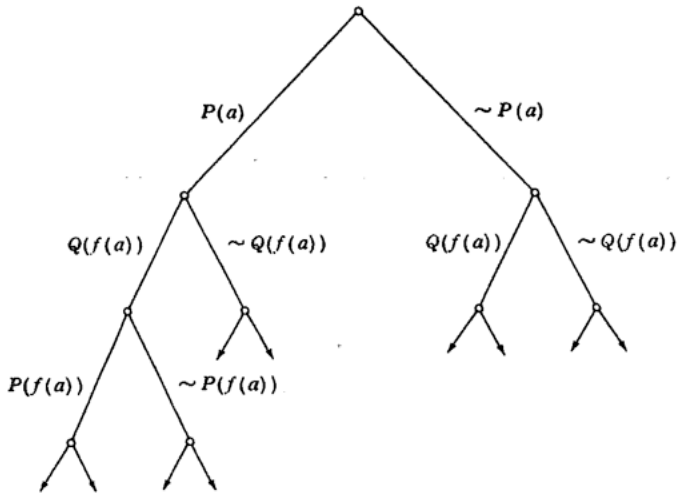


Figure 4.3

tree for S corresponds to an exhaustive survey of all possible interpretations for S. If S is unsatisfiable, then S fails to be true in each of these interpretations. Thus, we may stop expanding nodes from a node N if $I(N)$ falsifies S. This motivates the following definitions.

Definition A node N is a *failure node* if $I(N)$ falsifies some ground instance of a clause in S, but $I(N')$ does not falsify any ground instance of a clause in S for every ancestor node N' of N.

Definition A semantic tree T is said to be *closed* if and only if every branch of T terminates at a failure node.

Definition A node N of a closed semantic tree is called an *inference node* if all the immediate descendant nodes of N are failure nodes.

Example 4.13

Let $S = \{P, Q \vee R, \sim P \vee \sim Q, \sim P \vee \sim R\}$. The atom set of S is $A = \{P, Q, R\}$. Figure 4.4a is a complete semantic tree for S, while Fig. 4.4b is a closed semantic tree for S.

Example 4.14

Consider $S = \{P(x), \sim P(x) \vee Q(f(x)), \sim Q(f(a))\}$. The atom set of S is

$$A = \{P(a), Q(a), P(f(a)), Q(f(a)), \ldots\}.$$

Figure 4.5 shows a closed semantic tree for S.

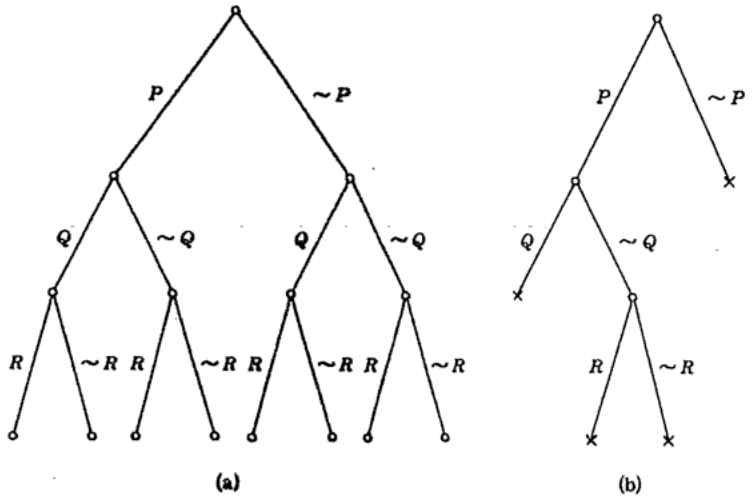


Figure 4.4

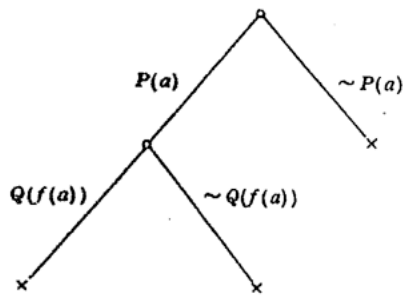


Figure 4.5

4.5 HERBRAND'S THEOREM

Herbrand's theorem is a very important theorem in symbolic logic; it is a base for most modern proof procedures in mechanical theorem proving. Herbrand's theorem is closely related to Theorem 4.2 given in Section 4.3. That is, to test whether a set S of clauses is unsatisfiable, we need consider only interpretations over the Herbrand universe of S. If S is false under all interpretations over the Herbrand universe of S, then we can conclude that S is unsatisfiable. Since there are usually many, possibly an infinite number, of these interpretations, we should organize them in some systematic way.

This can be done by using a semantic tree. We shall give two versions of Herbrand's theorem. The one stated most often in the literature is the second version; however, the first version is useful in this book.

Theorem 4.3 (Herbrand's Theorem, Version I) A set S of clauses is unsatisfiable if and only if corresponding to every complete semantic tree of S, there is a finite closed semantic tree.

Proof ($\Rightarrow$) Suppose S is unsatisfiable. Let T be a complete semantic tree for S. For each branch B of T, let I_B be the set of all literals attached to all links of the branch B. Then I_B is an interpretation for S. Since S is unsatisfiable, I_B must falsify a ground instance C' of a clause C in S. However, since C' is finite, there must exist a failure node N_B (which is a finite number of links away from the root node) on the branch B. Since every branch of T has a failure node, there is a closed semantic tree T' for S. Furthermore, since only a finite number of links are connected to each node of T', T' must be finite (that is, the number of nodes in T' is finite), for otherwise, by Konig's lemma [Knuth, 1968], we could find an infinite branch containing no failure node. Thus we complete the proof of the first half of the theorem.

($\Leftarrow$) Conversely, if corresponding to every complete semantic tree T for S there is a finite closed semantic tree, then every branch of T contains a failure node. This means that every interpretation falsifies S. Hence S is unsatisfiable. This completes the proof of the second half of the theorem.

Theorem 4.4 (Herbrand's Theorem, Version II). A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.

Proof ($\Rightarrow$) Suppose S is unsatisfiable. Let T be a complete semantic tree for S. Then, by Herbrand's theorem (version I), there is a finite closed semantic tree T' corresponding to T. Let S' be the set of all the ground instances of clauses that are falsified at all the failure nodes of T'. S' is finite since there are a finite number of failure nodes in T'. Since S' is false in every interpretation of S', S' is unsatisfiable.

($\Leftarrow$) Suppose there is a finite unsatisfiable set S' of ground instances of clauses in S. Since every interpretation I of S contains an interpretation I' of S', if I' falsifies S', then I must also falsify S'. However, S' is falsified by every interpretation I'. Consequently, S' is falsified by every interpretation I of S. Therefore, S is falsified by every interpretation of S. Hence, S is unsatisfiable. Q.E.D.

Example 4.15

Let $S = \{P(x), \sim P(f(a))\}$. This set S is unsatisfiable. Hence, by Herbrand's theorem, there is a finite unsatisfiable set S' of ground instances of clauses in S. We have found that one of these sets is $S' = \{P(f(a)), \sim P(f(a))\}$.

Example 4.16

Let $S = \{\sim P(x) \vee Q(f(x), x), P(g(b)), \sim Q(y, z)\}$. This set S is unsatisfiable. One of the unsatisfiable sets of ground instances of clauses in S is

$$S' = \{\sim P(g(b)) \vee Q(f(g(b)), g(b)), P(g(b)), \sim Q(f(g(b)), g(b))\}.$$

Example 4.17

Let the set S consist of the following clauses:

$$\begin{aligned} S = \{&\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(x, v, w) \vee P(u, z, w), \\ &\sim P(x, y, u) \vee \sim P(y, z, v) \vee \sim P(u, z, w) \vee P(x, v, w), \\ &P(g(x, y), x, y), P(x, h(x, y), y), P(x, y, f(x, y)), \\ &\sim P(k(x), x, k(x))\}. \end{aligned}$$

This set S is also unsatisfiable. However, it is not very easy to find by hand a finite unsatisfiable set S' of ground instances of clauses in S. One way to find such a set S' is to generate a closed semantic tree T' for S. Then the set S' of all the ground instances falsified at all the failure nodes of T' is such a desired set. The following is a desired set S'. The reader may want to check that each ground clause in S' is a ground instance of some clause in S, and that S' is unsatisfiable.

$$\begin{aligned} S' = \{&P(a, h(a, a), a), \\ &\sim P(k(h(a, a)), h(a, a), k(h(a, a))), \\ &P(g(a, k(h(a, a))), a, k(h(a, a))), \\ &\sim P(g(a, k(h(a, a))), a, k(h(a, a))) \vee \sim P(a, h(a, a), a) \\ &\vee \sim P(g(a, k(h(a, a))), a, k(h(a, a))) \vee P(k(h(a, a)), h(a, a), k(h(a, a)))\}. \end{aligned}$$

4.6 IMPLEMENTATION OF HERBRAND'S THEOREM

The second version of Herbrand's theorem suggests a refutation procedure. That is, given an unsatisfiable set S of clauses to prove, if there is a mechanical procedure that can successively generate sets $S_1', \ldots, S_n', \ldots$ of ground instances of clauses in S and can successively test $S_1', S_2', \ldots$ for unsatisfiability, then, as guaranteed by Herbrand's theorem, this procedure can detect a finite N such that S_N' is unsatisfiable.

Gilmore was one of the first men to implement the above idea [Gilmore, 1960]. In 1960, he wrote a computer program that successively generated sets $S_0', S_1', \ldots$, where S_i' is the set of all the ground instances obtained by replacing the variables in S by the constants in the i-level constant set H_i of S. Since each S_i' is a conjunction of ground clauses, one can use any method

available in the propositional logic to check its unsatisfiability. Gilmore used the multiplication method. That is, as each S_i' is produced, S_i' is multiplied out into a disjunctive normal form. Any conjunction in the disjunctive normal form containing a complementary pair is removed. Should some S_i' be empty, then S_i' is unsatisfiable and a proof is found.

Example 4.18

Consider

$$S = \{P(x), \sim P(a)\}.$$

$$H_0 = \{a\}$$

$$S_0' = P(a) \wedge \sim P(a) = \square.$$

Thus S is proved to be unsatisfiable.

Example 4.19

Consider

$$S = \{P(a), \sim P(x) \vee Q(f(x)), \sim Q(f(a))\}.$$

$$H_0 = \{a\}.$$

$$\begin{aligned} S_0' &= P(a) \wedge (\sim P(a) \vee Q(f(a))) \wedge \sim Q(f(a)) \\ &= (P(a) \wedge \sim P(a) \wedge \sim Q(f(a))) \vee (P(a) \wedge Q(f(a)) \wedge \sim Q(f(a))) \\ &= \square \vee \square = \square. \end{aligned}$$

Thus S is proved to be unsatisfiable.

The multiplication method used by Gilmore is inefficient. As is easily seen, even for a small set of ten two-literal ground clauses, there are 2^{10} conjunctions. To overcome this inefficiency, Davis and Putnam [1960] introduced a more efficient method for testing the unsatisfiability of a set of ground clauses. We shall now describe their method with some modification.

The Method of Davis and Putnam

Let S be a set of ground clauses. Essentially, the method consists of the following four rules.

I. *Tautology Rule* Delete all the ground clauses from S that are tautologies. The remaining set S' is unsatisfiable if and only if S is.

II. *One-Literal Rule* If there is a unit ground clause L in S, obtain S' from S by deleting those ground clauses in S containing L. If S' is empty,

S is satisfiable. Otherwise, obtain a set S'' from S' by deleting $\sim L$ from S'. S'' is unsatisfiable if and only if S is. Note that if $\sim L$ is a ground unit clause, then the clause becomes $\square$ when $\sim L$ is deleted from the clause.

III. *Pure-Literal Rule* A literal L in a ground clause of S is said to be *pure* in S if and only if the literal $\sim L$ does not appear in any ground clause in S. If a literal L is pure in S, delete all the ground clauses containing L. The remaining set S' is unsatisfiable if and only if S is.

IV. *Splitting Rule* If the set S can be put into the form

$$(A_1 \vee L) \wedge \cdots \wedge (A_m \vee L) \wedge (B_1 \vee \sim L) \wedge \cdots \wedge (B_n \vee \sim L) \wedge R,$$

where A_i, B_i, and R are free of L and $\sim L$, then obtain the sets $S_1 = A_1 \wedge \cdots \wedge A_m \wedge R$ and $S_2 = B_1 \wedge \cdots \wedge B_n \wedge R$. S is unsatisfiable if and only if $(S_1 \vee S_2)$ is unsatisfiable, that is, both S_1 and S_2 are unsatisfiable.

We can now show that the above rules are sound. That is, if the original set S is unsatisfiable, then the remaining set after one of the rules is applied is still unsatisfiable, and vice versa.

For Rule I Since a tautology is satisfied by every interpretation, S' is unsatisfiable if and only if S is.

For Rule II If S' is empty, then all the ground clauses in S contain L. Hence any interpretation containing L can satisfy S. Therefore S is satisfiable. We still have to show that S'' is unsatisfiable if and only if S is unsatisfiable. Suppose S'' is unsatisfiable. If S is satisfiable, then there is a model M of S containing L. For S'', M must satisfy all the clauses which do not contain L. Furthermore, since M falsifies $\sim L$, M must satisfy all the clauses that originally contain $\sim L$. Therefore, M must satisfy S''. This contradicts the assumption that S'' is unsatisfiable. Hence, S must be unsatisfiable. Conversely, suppose S is unsatisfiable. If S'' is satisfiable, then there is a model M'' of S''. Thus any interpretation of S containing M'' and L must be a model of S. This contradicts the assumption that S has no model. Hence S'' must be unsatisfiable. Therefore, S'' is unsatisfiable if and only if S is.

For Rule III Suppose S' is unsatisfiable. Then S must be unsatisfiable since S' is a subset of S. Conversely, suppose S is unsatisfiable. If S' is satisfiable, then there is a model M of S'. Since neither L nor $\sim L$ is in S', neither L nor $\sim L$ is in M. Thus any interpretation of S that contains M and L is a model of S. This contradicts the assumption that S has no model. Hence S' must be unsatisfiable. Therefore, S' is unsatisfiable if and only if S is unsatisfiable.

For Rule IV Suppose S is unsatisfiable. If $(S_1 \vee S_2)$ is satisfiable, then either S_1 or S_2 has a model. If $S_1(S_2)$ has a model M, then any interpretation

of S containing $\sim L(L)$ is a model of S. This contradicts the assumption that S has no model. Hence $(S_1 \vee S_2)$ is unsatisfiable. Conversely, suppose $(S_1 \vee S_2)$ is unsatisfiable. If S is satisfiable, S must have a model M. If M contains $\sim L(L)$, M can satisfy $S_1(S_2)$. This contradicts the assumption that $(S_1 \vee S_2)$ is unsatisfiable. Hence S must be unsatisfiable. Therefore, S is unsatisfiable if and only if $(S_1 \vee S_2)$ is.

The above rules are all very important. We shall see in the subsequent chapters that these rules have many extensions. We now give some examples to show how these rules can be used.

Example 4.20

Show that $S = (P \vee Q \vee \sim R) \wedge (P \vee \sim Q) \wedge \sim P \wedge R \wedge U$ is unsatisfiable.

(1) $(P \vee Q \vee \sim R) \wedge (P \vee \sim Q) \wedge \sim P \wedge R \wedge U$

(2) $(Q \vee \sim R) \wedge (\sim Q) \wedge R \wedge U$ Rule II on $\sim P$

(3) $\sim R \wedge R \wedge U$ Rule II on $\sim Q$

(4) $\square \wedge U$ Rule II on $\sim R$.

Since the last formula contains the empty clause $\square$, S is unsatisfiable.

Example 4.21

Show that $S = (P \vee Q) \wedge \sim Q \wedge (\sim P \vee Q \vee \sim R)$ is satisfiable.

(1) $(P \vee Q) \wedge \sim Q \wedge (\sim P \vee Q \vee \sim R)$

(2) $P \wedge (\sim P \vee \sim R)$ Rule II on $\sim Q$

(3) $\sim R$ Rule II on P

(4) ■ Rule II on $\sim R$.

The last set is an empty set. Hence S is satisfiable.

Example 4.22

Show that $S = (P \vee \sim Q) \wedge (\sim P \vee Q) \wedge (Q \vee \sim R) \wedge (\sim Q \vee \sim R)$ is satisfiable.

(1) $(P \vee \sim Q) \wedge (\sim P \vee Q) \wedge (Q \vee \sim R) \wedge (\sim Q \vee \sim R)$

(2) $(\sim Q \wedge (Q \vee \sim R) \wedge (\sim Q \vee \sim R))$
$\vee (Q \wedge (Q \vee \sim R) \wedge (\sim Q \vee \sim R))$ Rule IV on P

(3) $\sim R \vee \sim R$ Rule II on $\sim Q$ and Q

(4) ■ $\vee$ ■ Rule II on $\sim R$.

Since both of the split sets are satisfiable, S is satisfiable.

and in the above, call the second x, y say. This would mean that when we looked at such formulae as:

$$P(x) \wedge \neg P(f(f(f(f(f(y))))))$$

we could see straightaway that there was a potential match of $P(x)$ and $\neg P(f(f(f(f(f(y))))))$, by considering whether and how they would unify. So it is important to complete the clausal forming by standardising apart variables in different clauses sharing the same name.

Resolution makes use of this. Effectively it postpones tha actual generation of the Herbrand universe and base as long as possible, so that you only generate what you need.

4 Resolution

4.1 Resolution for Propositions

This is a partial development of resolution from Herbrand's theorem, see Alan Bundy's book for a more detailed account.

First, I want to look at the resolution rule of inference for propositions and then go back and look at it working with expressions involving predicates, so you can see it controlling the development of the Herbrand base.

If you think about Davis and Putnam's second rule, it would take something like:

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

and work out that this was unsatisfiable iff Q was. For if P is $\top$, the (un)satisfiability is determined by the second clause, which is effectively $\bot \vee Q$, i.e. Q. If P is $\bot$, the whole thing's $\bot$ anyway.

More generally, what about:

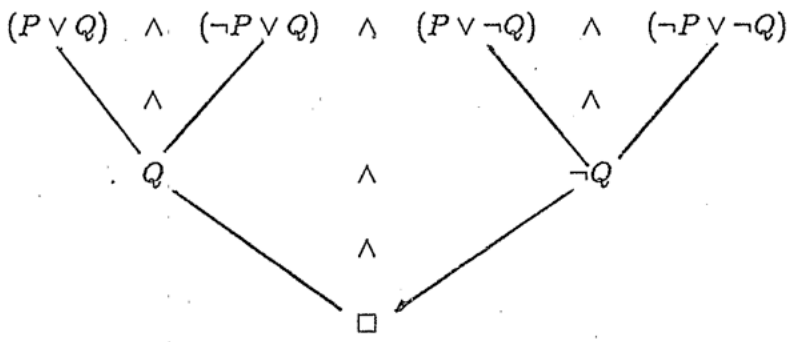
$$(P \vee R) \wedge (\neg P \vee Q)$$

Again there are two cases

- If P is $\top$ this is equivalent to $\top \wedge Q$, which is equivalent to Q.
- If P is $\bot$ this is equivalent to $R \wedge \top$, which is equivalent to R.

So effectively it's equivalent to $Q \vee R$. Having produced a resolvent, it should be ANDed back into the conjunction, available for further resolution with other clauses. If any resolution eventually produces the empty clause, e.g. $P \wedge \neg P$ resolving to $\Box$, that is unsatisfiable. So the basic process is to keep resolving until you get $\Box$.

Here's a resolution style deduction tree:



4.2 Resolution with Predicates

A rather more general formula than the one I used above to motivate using resolutionis:

$$(P(x) \vee Q(x)) \wedge (\neg P(f(y)) \vee R(y)) \wedge ...$$

The progressive Herbrand universe generation gives us something like this:

$$\begin{array}{ll} a & (P(a) \vee Q(a)) \wedge (\neg P(f(a)) \vee R(a)) \wedge ... \\ a, f(a) & (P(a) \vee Q(a)) \wedge (\neg P(f(a)) \vee R(a)) \wedge ... \\ & \wedge \\ & (P(f(a)) \vee Q(f(a))) \wedge (\neg P(f(f(a))) \vee R(f(a))) \wedge ... \end{array}$$

We would take $(P(f(a)) \vee Q(f(a)))$ and $(\neg P(f(a)) \vee R(a))$ and resolution would produce $Q(f(a)) \vee R(a)$, to AND back into the conjunction. But this is wasteful, we can use resolution without generating the Herbrand Universe on $(P(x) \vee Q(x))$ and $(\neg P(f(y)) \vee R(y))$ to produce $Q(f(z)) \vee R(z)$, knowing that the z can be instantiated by any member of the Herbrand Universe. The new clause is ANDed back into the conjunction, available for further resolutions. Its variables must be renamed apart from any existing ones.

So this is effectively genarating only those parts of the Herbrand Universe required to demonstrate unsatisfiability, and delaying such genarations as long as possible by unifying variables with variables to give other variables as long as possible. Obviously unifying a variable with a constant fixes the variable to that constant.

Search doesn't go away though. There is still the problem of choosing which clauses to resolve together.

Notice that if two literals can be unified there is a unique most general unifier.

5 Completeness and Soundness of Resolution

This is taken from Chang and Lee's account, and supplements the chapters on resolution, unification and formal results about resolution in Bundy's book.

5.1 Resolution Rule of Inference

Binary Resolution:

$$(C' \vee P') \wedge (C'' \vee \neg P'')$$

$$(C' \vee C'')\theta$$

where θ is the most general unifier of P' and P'', and the C's and P's stand for any literals.

Full Resolution:

$$(C' \vee P'_1 \vee ... \vee P'_n) \wedge (C'' \vee \neg P''_1 \vee ... \vee \neg P''_m)$$

$$(C' \vee C'')\phi$$

where $P'_1\phi \equiv ... \equiv P'_n\phi \equiv P''_1\phi \equiv ... \equiv P''_m\phi$ and ϕ is the most general such unifier.

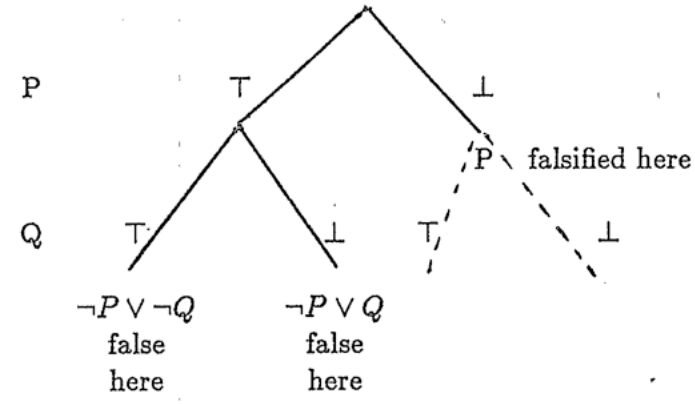
Resolution is used to derive new clauses which are ANDed to the existing ones once their variables have been renamed. In a refutation proof, this continues until the empty clause $\square$ is derived, or possibly forever if the original formula was satisfiable.

5.2 Resolution and Semantic Trees

Suppose we have some clauses on which resolution is being used:

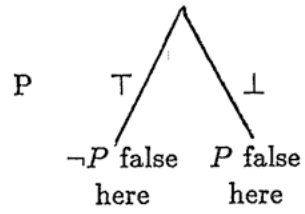
$$P \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$$

Here is the semantic tree:



$\neg P \vee \neg Q$ and $\neg P \vee Q$ are falsified at two nodes which, because they are children of a common parent, share a common assignment of truth values apart from the last assignment to Q. I.e. the two clauses contain the complementary literals Q and $\neg Q$. So they can be resolved together on these complementary literals. Each of the clauses was made false by the relevant assignments to Q, so everything else in the clauses $(\neg P)$ must already have been false. So since the resolvent of the two clauses is a disjunction of

everything else in them which was already false regardless of Q, it must be false in the sub-interpretation labelling the parent node. This new clause ANDed into the conjunction of clauses lets us prune the semantic tree, because we know that the parent of the two failure nodes we were considering before is a failure node for a clause in the conjunction (the new clause we just ANDed in). So we prune the branches below it, getting a new tree:



Another resolution in this case produces the empty clause from P and $\neg P$ and reduces the tree to just the root. We would not expect any assignment of truth values in an interpretation to be necessary to falsify the empty clause. If we reach the root, it means that the empty clause has been derived, since nothing else could be falsified by no assignment of truth values.

For the proofs that follow, we also need a lemma:

If C_1' is an instance of C_1 and C_2' is an instance of C_2, where C_1 and C_2 are clauses, and C' is a resolvent of C_1' and C_2' then $\exists$ C a resolvent of C_1 and C_2, such that C' is an instance of C

5.3 Completeness of Resolution

A conjunction of clauses S is unsatisfiable implies there is a deduction of the empty clause from S

Let $A_1, A_2, \ldots$ be S's Herbrand Base. S is unsatisfiable, so by Herbrand's theorem there is a finite closed semantic tree T' corresponding to every complete semantic tree. Take any such finite closed tree, then there are two cases:

- T' is only the root node. So the empty clause is already in S.
- T' is not only the root node. Then there must be a node N such that both its children N_1 and N_2 are failure nodes. These correspond to interpretations identical up to N and then with some element of the Herbrand Base (A_N say) true in N_1 and false in N_2. Since these are failure nodes, $\exists$ instances C_1' and C_2' of clauses C_1 and C_2, falsified by the interpretations at N_1 and N_2, but not by N's interpretation. Since the only development after N was the assignment of truth values to A_N, all the other literals in C_1' and C_2' were necessarily falsified at N, the only thing not making the whole clauses false was the lack of an assignment to A_N.

  So C_1' must contain $\neg A_N$ and C_2' must contain A_N. Resolve C_1' and C_2' on $\neg A_N, A_N$ to get C', which must be false in N's interpretation. From the lemma, we know there is some C, resolvent of C_1 and C_2 of which C' is an instance.

Let T'' be the tree pruned up to N. This is a closed semantic tree for $S \wedge C$, since an instance of C is falsified at N.

We can keep doing this, since the tree is finite, and we must eventually reach the empty (root only) tree, corresponding to the empty clause.

5.4 Soundness of Resolution

If there is a deduction of the empty clause from a conjunction of clauses S that implies S is unsatisfiable

Let $R_1, ..., R_k$ be the resolvents in the deduction. Suppose S were unsatisfiable, then it would have a model M. Any model must satisfy all S's clauses and all their resolvents, since the resolvents are logical consequences. It must also satisfy the resolvents' resolvents, and their reslovents etc. But $\square$ is such a resolvent, and nothing can be a model for it, so we have a contradiction, S cannot have been satisfiable.