6.4 RESOLUTION

We have argued that the inference rules covered so far are sound, but we have not discussed the question of **completeness**. Search algorithms are complete in the sense that they will find any reachable goal, but if the available inference rules are inadequate then a goal is not reachable, no proof exists which uses only those inference rules.

It would be useful from a computational point of view if we had a proof procedure that carried out in a single operation. The variety of process involved in reasoning with statements in predicate logic. Resolution is such a procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.

Resolution produces proof by **refutation**. In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statements. This approach contrasts with the technique that we have been using to generate proofs by chaining backward from the theorem to be proved to the axioms.

6.4.1 Conversion to Clause Form

Resolution requires that a set of **well formed formulas** (*wff*) to be converted to a set of clauses, where clauses is defined in conjunctive normal forms. The steps to convert wff to clause form are given below:

1. Eliminate $\rightarrow$ This is done by using fact that

   $$a \rightarrow b = \neg a \vee b$$

   Consider the following *wff;*

   $$\forall x: \forall y: ((have\ (x,y) \wedge cat\ (y)) \rightarrow (\neg\ \exists\ z:\ (have(x,z) \wedge mouse\ (z))))$$

   This wff means "*anyone who has any cats will not have any mice*". After elimination of $\rightarrow$ the result would be:

   $$\forall x: \forall y: ((\neg\ (have\ (x,y) \wedge cat\ (y)) \vee (\neg\ \exists z: (have(x,z) \wedge mouse\ (z))))$$

2. Move $\neg$ in as far as possible. Performing this operation results in:

   $$\forall x: \forall y: ((\neg\ have\ (x,y) \vee \neg\ cat\ (y)) \vee (\forall z: \neg\ have(x,z) \vee \neg\ mouse\ (z)))$$

   The result of $\neg \forall x:P(x)$ is equal to $\exists\ x:\neg P(x)$ and vice versa. The $\neg(a \wedge b)$ is equal to $\neg a \vee \neg b$ and vice versa. The single term $\neg(\neg a)$ is equal to a.

3. Move all quantifiers to the left without changing their relative order. This results in following *wff* called **prenex normal form.**

   $$\forall x: \forall y: \forall z: ((\neg\ have\ (x,y) \vee \neg\ cat\ (y)) \vee (\neg\ have(x,z) \vee \neg\ mouse\ (z)))$$

4. Eliminate $\exists$ existential quantifiers. (Skolemize)

   $$\forall x: \forall y: \forall z: ((\neg\ have\ (x,y) \vee \neg\ cat\ (y)) \vee (\neg\ have(x,z) \vee \neg\ mouse\ (z)))$$

In this example there is no existential quantifier. Every first-order formula can be converted into Skolem normal form while not changing its **satisfiability** via a process called **Skolemization**. The resulting formula is not necessarily **equivalent** to the original one, but is **equisatisfiable** with it i.e. it is satisfiable if and only if the original one is. The simplest form of Skolemization is for existentially quantified variables which are not inside the scope of a universal quantifiers. These can simply be replaced by creating new constants. For example, $\exists x: P(x)$ can be changed to $P(c)$, where c is a new constant.

More generally, Skolemization is performed by replacing every existentially quantified variable y with a term $f(x1, x2....xn)$ whose function symbol f is new. The variables of this term are as follows. If the formula is in prenex normal form, $x1, x2....xn$ are the variables that are universally quantified and whose quantifiers precede that of y. In general, they are the variables that are universally quantified and such that $\exists y$ occurs in the scope of their quantifiers. The function f introduced in this process is called a ***Skolem function*** (or **Skolem constant** if it is of zero arity) and the term is called a ***Skolem term.***

As an example, the formula $\forall x: \exists y: \forall z: P(x,y,z)$ is not in Skolem normal form because it contains the existential quantifier $\exists y$. Skolemization replaces y with $f(x)$, where f is a new function symbol and removes the quantification over y. The resulting formula is $\forall x: \forall z: P(x,f(x),z)$.

5. Eliminate $\forall$ because now all the variables are universally quantified.

$$(\neg \text{ have } (x,y) \vee \neg \text{ cat } (y)) \vee (\neg \text{ have}(x,z) \vee \neg \text{ mouse } (z))$$

6. Convert the formula into Conjunctive Normal Form (CNF) or Conjunction of disjuncts using the following two properties:

   1. **Associative Property :** $a \vee (b \vee c) = (a \vee b) \vee c$
   2. **Distributive Property :** $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

   For example, convert the following expression into CNF :

   $(Winter \wedge Shoes) \vee (Summer \wedge Sandals)$

   Applying the distributive rule, the formula becomes :

   $[Winter \vee (Summer \wedge Sandals)] \wedge [Shoes \vee (Summer \wedge Sandals)]$

   It can be written as;

   $(Winter \vee Summer) \wedge$

   $(Winter \vee Sandals) \wedge$

   $(Shoes \vee Summer) \wedge$

   $(Shoes \vee Sandals)$

7. Put into clausal form by separating each conjunct :

   1. *Winter* ∨ *Summer*
   2. *Winter* ∨ *Sandals*
   3. *Shoes* ∨ *Summer*
   4. *Shoes* ∨ *Sandals*

Finally, the example formula becomes:

$$\{\{\neg have(x,y)\}, \{\neg cat(y)\}, \{\neg have(x,z)\}, \{\neg mouse(z)\}\}$$

6.4.2 Resolution in Propositional Logic

Resolution in propositional logic is somewhat easier then resolution in predicate logic. First understand the resolution procedure for propositional logic. The algorithm in given below:

Algorithm: Propositional Logic

1. Convert all the propositions to clause form **(S)**.
2. Negate **α** and convert it to clause form. Add it to **S**.
3. Repeat until either a contradiction is found or no progress can be made:
   a. Select two clauses such that (**α** ∨ ¬P) and (**γ** ∨ P). Eliminate ¬P and P from the resolvent.
   b. Add the resolvent (**α** ∨ **γ**) to **S**, if resolvent is not empty.

Algorithm 6.1 : Resolution in Propositional Logic

Example 4. Understand the procedure with the help of example. The set of axioms are given below:

1. $(S \wedge W) \rightarrow Q$
2. $(R \vee P) \rightarrow W$
3. S
4. P

We have to prove Q.

Solution: Converting them to clause form, results in following axioms:

1. $\neg S \vee \neg W \vee Q$
2. $(\neg R \wedge \neg P) \vee W$. Converting into CNF, it becomes $\neg R \vee W$ and $\neg P \vee W$
3. S
4. P

There are 5 clauses

1. $\neg S \vee \neg W \vee Q$
2. $\neg R \vee W$
3. $\neg P \vee W$
4. S
5. P

The resolution procedure is started with one literal as $\neg Q$(negation of Q that is to be proved) and selecting one from list above having complementary literal which is clause 1. The further steps of resolution are shown in following figure 6.1:

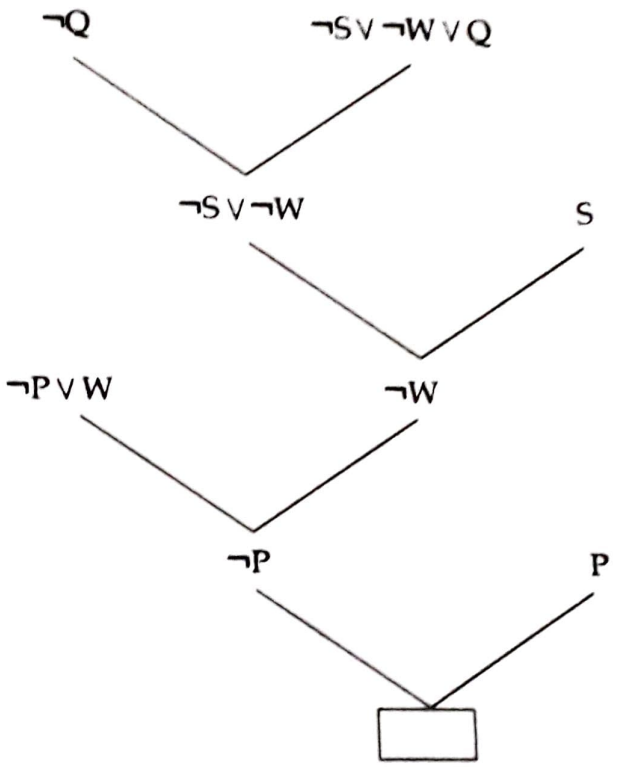

Figure 6.1 : Example of Resolution in Propositional Logic

Step 1: The clause 1 contradicts with $\neg Q$. The remaining resolvent is $\neg S \vee \neg W$.

Step 2: The proposition $\neg S \vee \neg W$ contradicts with S. The remaining resolvent is $\neg W$.

Step 3: The proposition $\neg W$ contradicts with $\neg P \vee W$. The remaining resolvent is $\neg P$.

Step 4: The proposition $\neg P$ contradicts with P. The resolvent is empty clause. Contradiction has been found.

Example 5. Use proposition logic to test the validity of the following argument: **[Raj.Univ. 2002]**

- If I study, then I will not fail in maths.
- If I do not play basketball, then I will study. But I failed in maths.

Conclusion: Therefore I must have played basketball.

Solution: Represent the above arguments in propositional form:

1. If I study. Let's say it P.
2. I will not fail in maths. Let's say it Q.
3. If I do not play basketball. Let's say it R.
4. But I failed in maths. Let's say it $\neg Q$.
5. Therefore I must have played basketball. Let's say it $\neg R$.

Thus, above statements in propositional logic form are:

1. $P \rightarrow Q$
2. $R \rightarrow P$
3. $\neg Q$

Conclusion: $\neg R$

Converting them to clause form, results in following axioms :

1. $\neg P \vee Q$
2. $\neg R \vee P$
3. $\neg Q$
4. R

The resolution procedure is started with one literal as R(negation of R that is to be proved) and selecting one from list above having complementary literal which is clause 2. The further steps of resolution are shown in following figure 6.2:

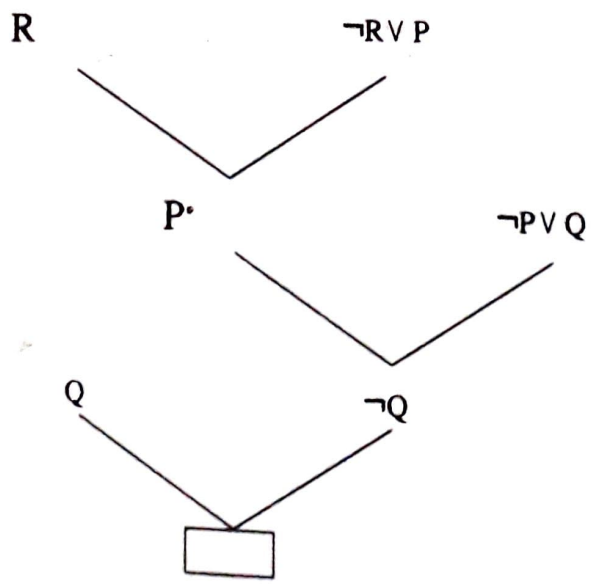

Figure 6.2 : Example of Resolution in Propositional Logic

The resolvent is empty clause. Contradiction has been found, therefore I must have played basketball. The validity of the statement can also be proved by truth table. Try Yourself !!

6.4.3 Unification Algorithm

The process of finding the substitution is **unification**. In propositional logic, we have seen that two literals (such as p and $\neg p$) contradict with each other. In FOPL, this matching process is more complicated because of arguments. For example, *girl(Jimmy)* and ¬*girl(Jimmy)* is a contradiction, while *girl(Jimmy)* and *girl(rose)* is not.

To detect contradictions in FOPL, we need a matching procedure that compares two literals and discovers whether there exists a set of **substitutions** that makes them identical. This procedure is called **unification**. The substitution itself is a **unifier**. The process helps focus search on substitutions that matter. Unify takes two atomic sentences p and q and returns a substitution that makes them look the same. The literals with different predicate symbols cannot be unified. For example following literals cannot be unified.

girl(Jimmy)

boy(Jimmy)

Match the arguments of the predicates. Identical predicates and constants can be matched. The unification algorithm takes two atomic literals, such as *Knows(John, x)* and *Knows(John, Paul)*, and returns a substitution that makes them look the same, such as {x/Paul}. If there is no such substitution, the unification fails. The notation *x / Paul* means *Paul* is substituted for *x*.

There may be more than one substitution that unifies two clauses. In fact, there may be infinitely many. The unification algorithm returns the "most general unifier", that is, the substitution that makes the least commitment about the bindings of variables.

The Unification Algorithm:

Unification performs *syntactic pattern matching*. Therefore the algorithm doesn't care about the difference between predicates, constants, variables, and functions. It only cares about symbols.

To simplify the algorithm we will therefore use list notation for our expressions, whereby a predicate name or function name is placed together with the predicate's or function's parameters in a single list:

CONVENTIONAL NOTATION	LIST NOTATION
p(a,b)	*(p a b)*
p(f(a),g(X,Y))	*(p (f a) (g X Y))*
equal(eve,mother(cain))	**(equal eve (mother cain))**

```
ALGORITHM UNIFY(E1, E2)
begin
  case
   both E1 and E2 are constants or the empty list:  % base case
       if E1 = E2
           then return {}
       else return FAIL;

  E1 is a variable:
       if E1 occurs in E2
           then return FAIL;
       else return {E2/E1}

  E2 is a variable:
       if E2 occurs in E1
           then return FAIL;
       else return {E1/E2}
  otherwise:                                  % both E1 and E2 are lists
       begin
        HE1 := first element of E1;
        HE2 := first element of E2;
        SUBS1 := unify(HE1, HE2);             %   recursion
        if SUBS1 = FAIL, then return FAIL;
        TE1 := apply(SUBS1, rest of E1);
        TE2 := apply(SUBS1, rest of E2);
        SUBS2 := unify(TE1, TE2);             %  recursion
        if SUBS2 = FAIL then return FAIL;
          else return composition(SUBS1, SUBS2)
       end
end
```

Algorithm 6.2 : Unification Algorithm

Let us look some examples of how unification procedure work. Suppose we have query *Knows(John,x)*: whom does John know? Some answers to this query can be found by finding all sentences in the knowledge base that unify with *Knows(John,x)*. Here are the results of unification with four different sentences that might be in the knowledge base.

UNIFY (*Knows(John,x), Knows(John,Jane)*) = *{x/Jane}*

UNIFY (*Knows(John,x), Knows(y,Bill)*) = *{x/Bill, y/John}*

UNIFY (*Knows(John,x), Knows(y,Mother(y))*) = *{y/John, x/Mother(John)}*

UNIFY (*Knows(John,x), Knows(x,Elizabeth)*) = *{FAIL}*

The last unification fails because x cannot take on the values ***John*** and ***Elizabeth*** at the same time. The problem can be avoided by standardizing apart one of the two sentences being unified, which means renaming its variables to avoid clashes. For example, we can rename x in *Knows(x, Elizabeth)* to z without changing its meaning. Now unification will work:

UNIFY (*Knows(John, x), Knows(z, Elizabeth)* = *{x/ Elizabeth, z/ John}*

6.4.4 Resolution in Predicate Logic

The resolution procedure followed in predicate logic is similar to propositional logic except the additional function of unification. The algorithm in given below:

Algorithm: Predicate Logic

1. Convert all the statements to clause form **(S)**.
2. Negate α and convert it to clause form. Add it to **S**.
3. Repeat until either a contradiction is found or no progress can be made.
   a. Select two clauses such that $(\alpha \vee \neg P)$ and $(\gamma \vee P)$. Eliminate $\neg P$ and P from the resolvent. Use unification algorithm to resolve the complementary literals, if required.
   b. Add the resolvent $(\alpha \vee \gamma)$ to S, if resolvent is not empty.

Algorithm 6.3 : Resolution in Predicate Logic

Example 6. Consider the following sentences:

1. Vicky likes all kinds of food.
2. Potato is food.
3. Carrot is food.
4. Anything anyone eats and is not killed by is food.
5. Brown eats chocolate and is still alive.
6. Cris eats everything Brown eats.

Perform the following operations on given set of statements:

(a) Translate these sentences into predicate logic

(b) Convert the formulas of part (a) into clause form.

(c) Prove that "Vicky likes chocolate" using resolution.

(d) Answer the question, "What food does Cris eat?"

Solution:

(a) The predicate logic representation of above sentences are:

(i) $\forall x$: *food (x)* $\rightarrow$ *likes (vicky, x)*

(ii) *food (potato)*

(iii) *food (carrot)*

(iv) $\forall x$: $\exists y$: *eat (y,x)* $\wedge$ $\neg$*killed (y)* $\rightarrow$ *food (x)*

(v) *eat (Brown, chocolate)* $\wedge$ $\neg$*killed (Brown)*

(vi) $\forall x$: *food (x)* $\wedge$ *eat (Brown, x)* $\rightarrow$ *eat (Cris, x)*

(b) The clausal form of above *wff* are:

(i) $\neg$ *food (x)* $\vee$ *likes (vicky, x)*

(ii) *food (potato)*

(iii) *food (carrot)*

(iv) $\neg$ *[eat (y, x)* $\wedge$ $\neg$*killed (y)]* $\vee$ *food (x)*

$\neg$ *eat (y, x)* $\vee$ *killed (y)* $\vee$ *food (x)*

(v) *eat (Brown, chocolate)* $\vee$ $\neg$ *killed (Brown)*

There are two clauses:

(a) eat (Brown, chocolate)

(b) $\neg$*killed (Brown)*

(vi) $\neg$*[food (x)* $\wedge$ *eat (Brown, x)]* $\vee$ *eat (Cris, x)*

$\neg$ *food (x)* $\vee$ $\neg$*eat (Brown, x)* $\vee$ *eat (Cris, x)*

(c) Prove: *likes (Vicky, chocolate)*

Negating the formula and add to set of clauses

$\neg$*likes (Vicky, chocolate)*

The figure 6.3 shows the resolution steps.

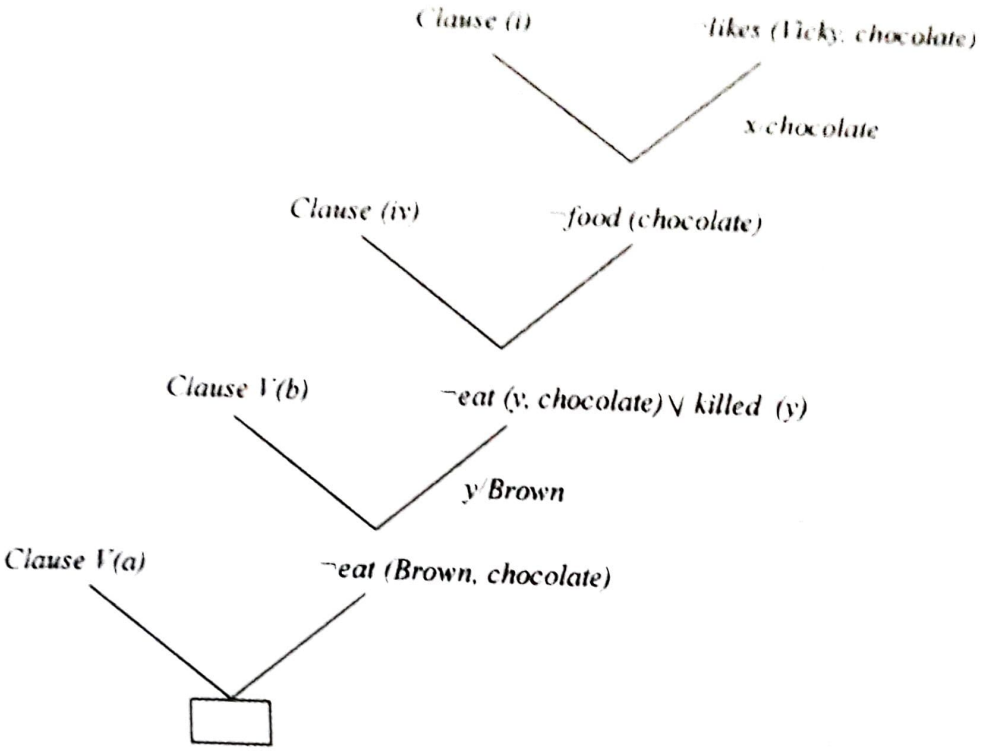

Figure 6.3 : Proof of likes (Vicky, chocolate)

Contradiction is found, hence it is proved that "Vicky likes chocolate".

(d) It is given that, "Cris eats everything Brown eats", and from resolution we know that Brown likes chocolate. So, we can say Cris also eats chocolate.

Example 7. Consider the following facts: **[Raj. Univ. 2006, 2009]**

1. Peter only likes easy courses.
2. Mathematics courses are hard.
3. All the courses in the textile department are easy.
4. TXT281 is a textile course.

Answer the question, "What course would Peter like?"

Solution: We will answer this question with the help of resolution, and prove: *likes (Peter, x)*

Convert the given facts into predicate logic. These are:

(i) $\forall x$: *course (x)* $\wedge$ *easy (x)* $\rightarrow$ *likes (Peter, x)*

(ii) *hard (Mathematics)*

(iii) $\forall x$: *course (x)* $\wedge$ *department (textile)* $\rightarrow$ *easy (x)*

(iv) *course (TXT281)* $\wedge$ *department (textile)*

The clause form of these sentences are:

(i) $\neg [course(x) \wedge easy(x)] \vee likes(Peter, x)$

$\neg course(x) \vee \neg easy(x) \vee likes(Peter, x)$

(ii) $hard(Mathematics)$

(iii) $\neg [course(x) \wedge department(textile)] \vee easy(x)$

$\neg course(x) \vee \neg department(textile) \vee easy(x)$

(iv) (a) $course(TXT281)$

(b) $department(textile)$

Add the negation of $likes(Peter, x)$ i.e. $\neg likes(Peter, x)$ into clause set. The figure 6.4 shows the resolution procedure.

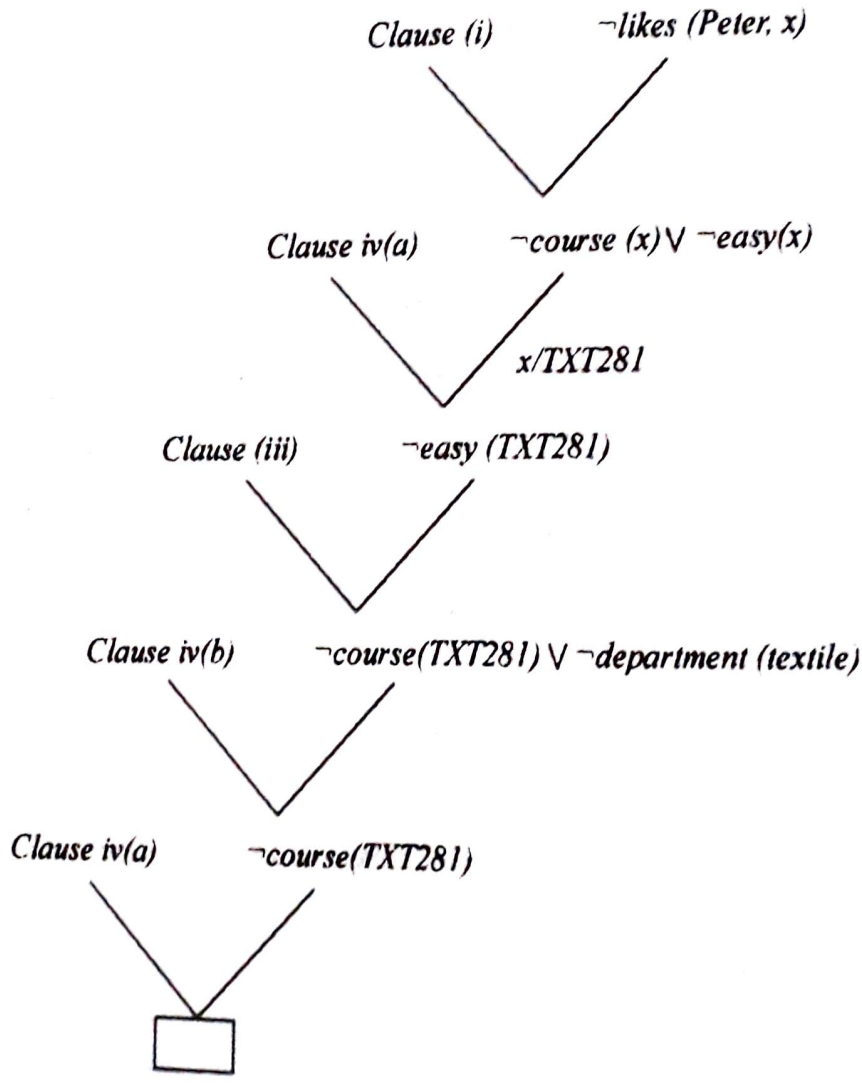

Figure 6.4 : Proof of likes (Peter, x)

From this resolution procedure, we came to know that Peter likes TXT281 course.

Example 8. The set of 4 statements are given below:

1. Anyone passing his history exams and winning the lottery is happy.
2. But anyone who studies or is lucky can pass all his exams.
3. John did not study but John is lucky.
4. Anyone who is lucky wins the lottery.

Prove: Is John happy?

Solution: The predicate logic representation of these statements are:

1. $\forall x: Pass(x, History) \wedge Win(x, Lottery) \rightarrow Happy(x)$
2. $\forall x: \forall y: Study(x) \vee Lucky(x) \rightarrow Pass(x,y)$
3. $\neg Study(John) \wedge Lucky(John)$
4. $\forall x: Lucky(x) \rightarrow Win(x, Lottery)$

Converting them to clause form:

1. $\forall x: Pass(x, History) \wedge Win(x, Lottery) \rightarrow Happy(x)$

   $\forall x: \neg(Pass(x, History) \wedge Win(x, Lottery)) \vee Happy(x)$

   $\forall x: \neg Pass(x, History) \vee \neg Win(x, Lottery) \vee Happy(x)$

   $\neg Pass(x, History) \vee \neg Win(x, Lottery) \vee Happy(x)$

2. $\forall x: \forall y: Study(x) \vee Lucky(x) \rightarrow Pass(x,y)$

   $\forall x: \forall y: \neg(Study(x) \vee Lucky(x)) \vee Pass(x,y)$

   $\forall x: \forall y: \neg Study(x) \wedge \neg Lucky(x) \vee Pass(x,y)$

   $(\neg Study(x) \wedge \neg Lucky(x)) \vee Pass(x,y)$

   $(\neg Study(x) \vee Pass(x,y)) \wedge (\neg Lucky(x) \vee Pass(x,y))$

   a. $\neg Study(x) \vee Pass(x,y)$

   b. $\neg Lucky(x) \vee Pass(x,y)$

3. $\neg Study(John) \wedge Lucky(John)$

   a. $\neg Study(John)$

   b. $Lucky(John)$

4. $\forall x: Lucky(x) \rightarrow Win(x, Lottery)$

   $\forall x: \neg Lucky(x) \vee Win(x, Lottery)$

   $\neg Lucky(x) \vee Win(x, Lottery)$

The goal is to prove *Happy(John)*. Add the negation ¬*Happy(John)* to the set of clauses. Resolve the set of clauses until FALSE is derived. The resolution steps are shown in figure 6.5.

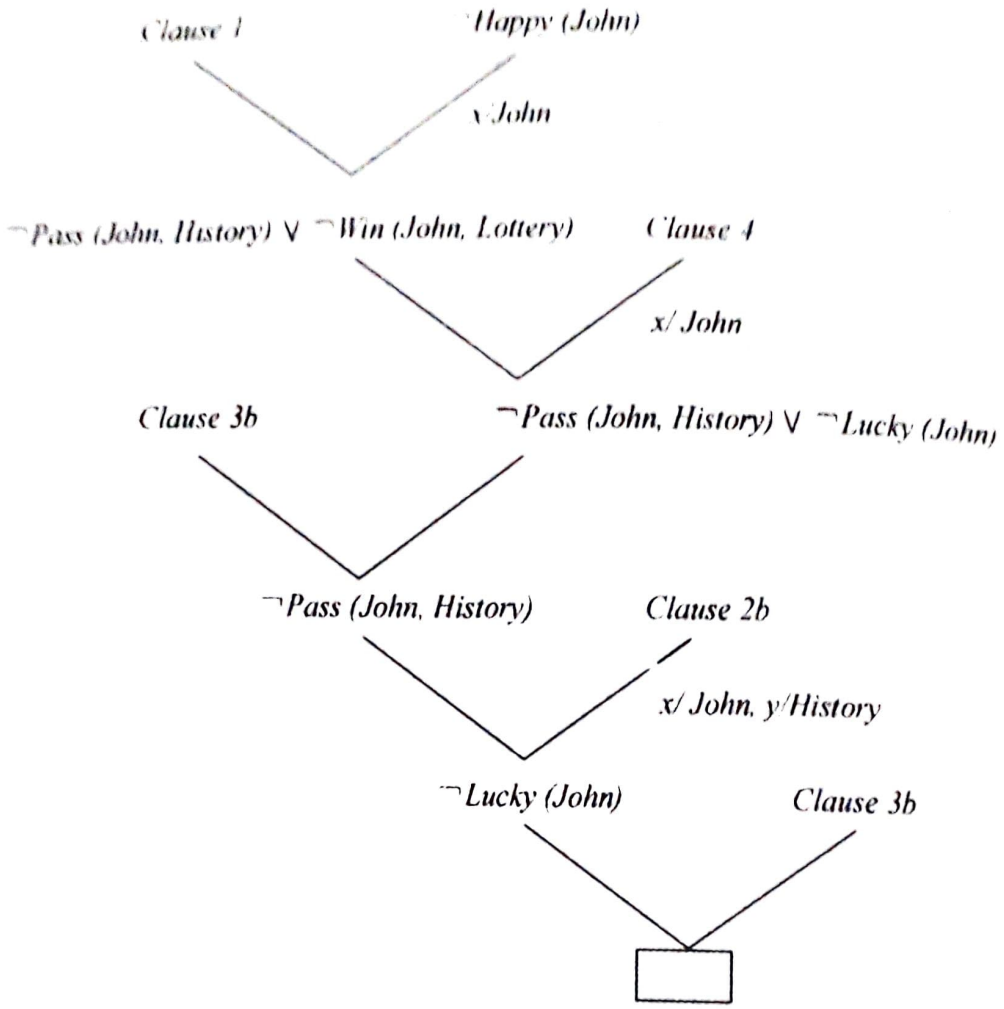

Figure 6.5 : Example of Resolution in Predicate Logic

In the clause ¬ *Pass(x, History)*, *x* is substituted with *John*, to resolve the expression. As we see contradiction is found at the final step, so it is proved that *John* is *happy*.

Example 9. Consider the following facts :

1. All hounds howl at night.
2. Anyone who has any cats will not have any mice.
3. Light sleepers do not have anything which howls at night.
4. John has either a cat or a hound.
5. (Conclusion) If John is a light sleeper, then John does not have any mice.

Solution: The conclusion can be proved using Resolution. The first step is to write each axiom as a well-formed formula in first-order predicate calculus. The clauses written for the above axioms are shown below, using *LS(x)* for 'light sleeper'.