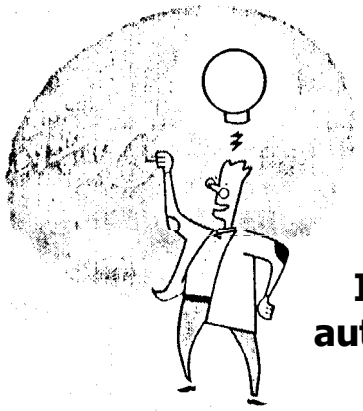
CHAPTER 3



RESOLUTION THEOREM PROVING

In this chapter we present a technique for automating the process of finding solutions to user's query using resolution.

By: Muhanad Tahrir Younis

3-1 What is Resolution ?

Resolution is a technique for theorem proving in propositional and predicate calculus which attempts to show that the negation of the statement produces a contradiction with the known statements.

Resolution refutation proofs involve the following steps:

1. Put the premises or axioms into clause form.
2. Add the negation of what is to be proved, in clause form, to the set of axioms.
3. Resolve these clauses together, producing new clauses that logically follow from them.
4. Produce a contradiction by generating the empty clause.
5. The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true.

Resolution refutation proofs require that the axioms and the negation of the goal be placed in a normal form called **clause form**. Clause form represents the logical database as a set of disjunctions of literals. A literal is an atomic expression or the negation of an atomic expression.

The most common form of resolution, called **binary resolution**, is applied to two clauses when one contains a literal and the other its negation. If these literals contain variables, the literals must be unified to make them equivalent. A new clause is then produced consisting of the disjuncts of all the predicates in the two clauses minus the literal and its negative instance, which are said to have been "resolved away." The resulting clause receives the unification substitution under which the predicate and its negation are found as "equivalent."

3-2 Producing the Clause Form for Resolution Refutations

The resolution proof procedure requires all statements in the database describing a situation to be converted to a standard form called **clause form**. This is motivated by the fact that resolution is an operator on pairs of disjuncts to produce new disjuncts. The form the database takes is referred to as a conjunction of disjuncts. It is a conjunction because all the clauses that make up the database are assumed to be true at the same time. It is a disjunction in that each of the individual clauses is expressed with disjunction (or v) as the connective.

We now present an algorithm, consisting of a sequence of transformations, for reducing any set of predicate calculus statements to clause form.

We demonstrate this process of conjunctive normal form reduction through an example and give a brief description rationalizing each step.

In the following expression, uppercase letters indicate variables (W, X, Y, and Z); lowercase letters in the middle of the alphabet indicate constants or bound variables (l, m, and n); and early alphabetic lowercase letters indicate the predicate names (a, b, c, d, and e). To improve readability of the expressions, we use two types of brackets: ( ) and [ ]. Where possible in the derivation, we remove redundant brackets: The expression we will reduce to clause form is:

(i) (∀X)([a(X) ∧ b(X)] ⇒ [c(X,l) ∧ (∃Y)((∃Z)[c(Y,Z)] ⇒ d(X,Y))]) v (∀X)(e(X))

1. First we eliminate the ⇒ by using: a ⇒ b ≡ ¬ a v b. This transformation reduces the expression in (i) above:

(ii) (∀X)(¬[a(X) ∧ b(X)] v [c(X,l) ∧ (∃Y)(¬(∃Z)[c(Y,Z)] v d(X,Y))]) v (∀X)(e(X))

2. Next we reduce the scope of negation. This may be accomplished using a number of the transformations that includes:

$$\neg(\neg a) \equiv a$$
$$\neg(\exists X)\, a(X) \equiv (\forall X) \neg a(X)$$
$$\neg(\forall X)\, b(X) \equiv (\exists X) \neg b(X)$$
$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$
$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

Using the second and fourth equivalences (ii) becomes:

(iii) (∀X)([¬ a(X) v¬b(X)] v [c(X,l) ∧ (∃Y)((∀Z)[¬c(Y,Z)] v d(X,Y))]) v (∀X)(e(X))

3. Next we standardize by renaming all variables so that variables bound

by different quantifiers have unique names. Because variable names are "dummies" or "place holders," the particular name chosen for a variable does not affect either the truth value or the generality of the clause. Transformations used at this step are of the form:

$$((\forall X)a(X) \vee (\forall X)b(X)) \equiv (\forall X)a(X) \vee (\forall Y)b(Y)$$

Because (iii) has two instances of the variable X, we rename:

(iv) (∀X)([¬ a(X) v ¬b(X)] v [c(X,I) ∧ (∃Y)((∀Z)[¬c(Y,Z)] v d(X,Y))]) v (∀W)(e(W))

4. Move all quantifiers to the left without changing their order. This is possible because step 3 has removed the possibility of any conflict between variable names. (iv) now becomes:

(v) (∀X)(∃Y)(∀Z)(∀W)([¬ a(X) v¬b(X)] v [c(X,I) ∧ ([¬c(Y,Z)] v d(X,Y))]) v e(W))

After step 4 the clause is said to be in **prenex** normal form, because all the quantifiers are in front as a **prefix** and the expression or **matrix** follows after.

5. At this point all existential quantifiers are eliminated by a process called *skolemization.* Expression (v) has an existential quantifier for Y. When an expression contains an existentially quantified variable, for example, (∃Z)(foo(.... Z,...)), it may be concluded that there is an assignment to Z under which foo is true. Skolemization identifies such a value. Skolemization does not necessarily show how to produce such a value; it is only a method for giving a name to an assignment that must exist. If k represents that assignment, then we have foo(.... k....). Thus:

(∃X)(dog (X)) may be replaced by dog(fido)

where the name fido is picked from the domain of definition of X to represent that individual X. fido is called a **skolem constant**. If the predicate has more than one argument and the existentially quantified variable is within the scope of universally quantified variables, the existential variable must be a function of those other variables. This is represented in the skolemization process:

$$(\forall X)\ (\exists Y)\ (\text{mother}\ (X,Y))$$

This expression indicates that every person has a mother. Every person is an X and the existing mother will be a function of the particular person X that is picked. Thus skolemization gives:

$$(\forall X)\text{mother}\ (X,\ m(X))$$

which indicates that each X has a mother (the m of that X). In another example:

$$(\forall X)(\forall Y)(\exists Z)(\forall W)(\text{foo}(X,Y,Z,W))$$

is skolemized to:

$$(\forall X)(\forall Y)(\forall W)(\text{foo}(X,Y,f(X,Y),W)).$$

We note that the existentially quantified Z was within the scope (to the right of) universally quantified X and Y Thus the skolem assignment is a function of X and Y but not of W. With skolemization (v) becomes:

(vi) $(\forall X)(\forall Z)(\forall W)([\neg\, a(X) \vee \neg b(X)] \vee [c(X,l) \wedge ([\neg c(f(X),Z)] \vee d(X,f(X)))]) \vee e(W))$

where f is the skolem function of X that replaces the existential Y. Once the skolemization has occurred, step 6 can take place, which simply drops the prefix.

6. Drop all universal quantification. By this point only universally quantified variables exist (step 5) with no variable conflicts (step 3). Thus all quantifiers can be dropped, and any proof procedure employed assumes all variables are universally quantified. Formula (vi) now becomes:

(vii) [¬ a(X) v ¬b(X)] v [c(X,l) ∧ (¬c(f(X),Z) v d(X,f(X)))] v e(W)

7. Next we convert the expression to the conjunct of disjuncts form. This requires using the associative and distributive properties of ∧ and v. Recall that

$$a \vee (b \vee c) = (a \vee b) \vee c$$
$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

which indicates that ∧ or v may be grouped in any desired fashion. The distributive property is also used, when necessary. Because

$$a \wedge (b \vee c)$$

is already in clause form, ∧ is not distributed. However, v must be distributed across ∧ using:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

The final form of (vii) is:

(viii) [¬ a(X) v ¬b(X) v c(X,l) v e(W)] ∧
[¬ a(X) v ¬b(X) v ¬c(f(X),Z) v d(X,f(X)) v e(W)]

8. Now call each conjunct a separate clause. In the example (viii) above there are two clauses:

(ixa) [¬ a(X) v ¬b(X) v c(X,l) v e(W)]
(ixb) [¬ a(X) v ¬b(X) v ¬c(f(X),Z) v d(X,f(X)) v e(W)]

9. The final step is to standardize the variables apart again. This requires giving the variable in each clause generated by step 8 different names. This procedure arises from the following equivalence:

$$(\forall X)\,(a(X) \wedge b(X)) \equiv (\forall X)\,a(X) \wedge (\forall Y)\,b(Y)$$

which follows from the nature of variable names as place holders. (ixa) and (ixb) now become, using new variable names U and V:

(xa) [¬ a(X) v ¬b(X) v c(X,l) v e(W)]
(xb) [¬ a(U) v ¬b(U) v ¬c(f(U),Z) v d(U,f(U)) v e(V)]

3-3 The Binary Resolution Proof Procedure

The resolution refutation proof procedure answers a query or deduces a new result by reducing the set of clauses to a contradiction, represented by the null clause (□). The contradiction is produced by resolving pairs of clauses from the database. If a resolution does not produce a contradiction directly, then the clause produced by the resolution, the resolvent, is added to the database of clauses and the process continues.

Ex(1): Consider now an example from the propositional calculus, where we want to prove a from the following axioms:

b **∧** c → a
b
d **∧** e → c
e **v** f
d **∧** ¬ f

We reduce the first axiom to clause form:

b ∧ c → a

¬ (b ∧ c) v a by $l \rightarrow m \equiv \neg l \vee m$

¬ b v ¬ c v a by de Morgan's law

The remaining axioms are reduced, and we have the following clauses:

a ∨ ¬ b ∨ ¬ c

b

c ∨ ¬ d ∨ ¬ e

e ∨ f

d

¬ f

The resolution proof is found in Figure (3-1). First, the goal to be proved, a, is negated and added to the clause set. The derivation of □ indicates that the database of clauses is inconsistent.

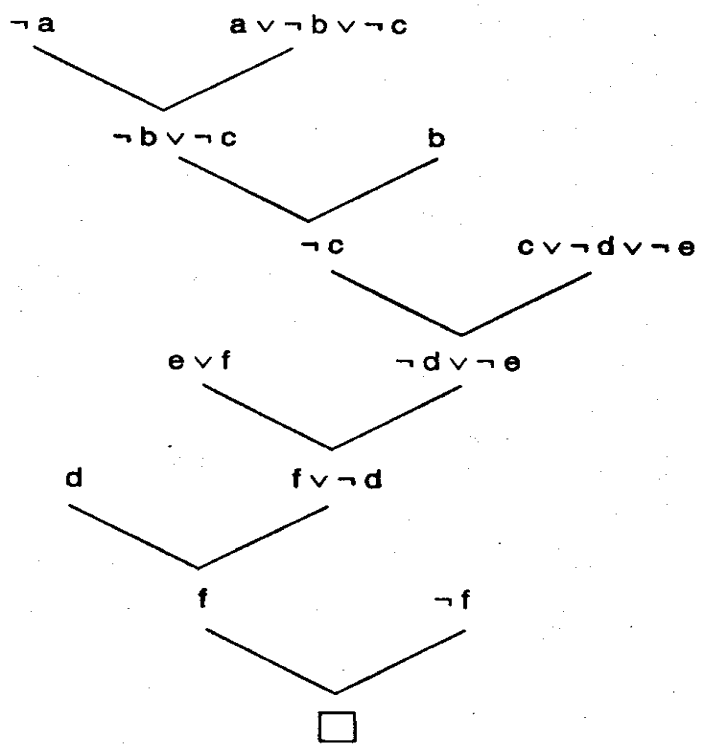


Figure (3-1): Resolution prove for an example from the propositional calculus.

Ex(2): We now present an example of a resolution refutation for the predicate calculus. We wish to prove that "Fido will die" from the statements that "Fido is a dog. All dogs are animals. All animals will die."

Fido is a dog: dog (fido).

All dogs are animals: $\forall(X)$ (dog (X) $\rightarrow$ animal (X)).

All animals will die: $\forall(Y)$ (animal (Y) $\rightarrow$ die (Y)).

Converts these predicates to clause form:

	PREDICATE FORM	CLAUSE FORM
1.	$\forall(X)$ (dog) (X) $\rightarrow$ animal (X))	$\neg$ dog (X) $\vee$ animal (X)
2.	dog (fido)	dog (fido)
3.	$\forall(Y)$ (animal (Y) $\rightarrow$ die (Y))	$\neg$ animal (Y) $\vee$ die (Y)

Negate the conclusion that Fido will die:

4.	$\neg$ die (fido)	$\neg$ die (fido)

Resolve clauses having opposite literals, producing new clauses by resolution as in Figure (3-2).

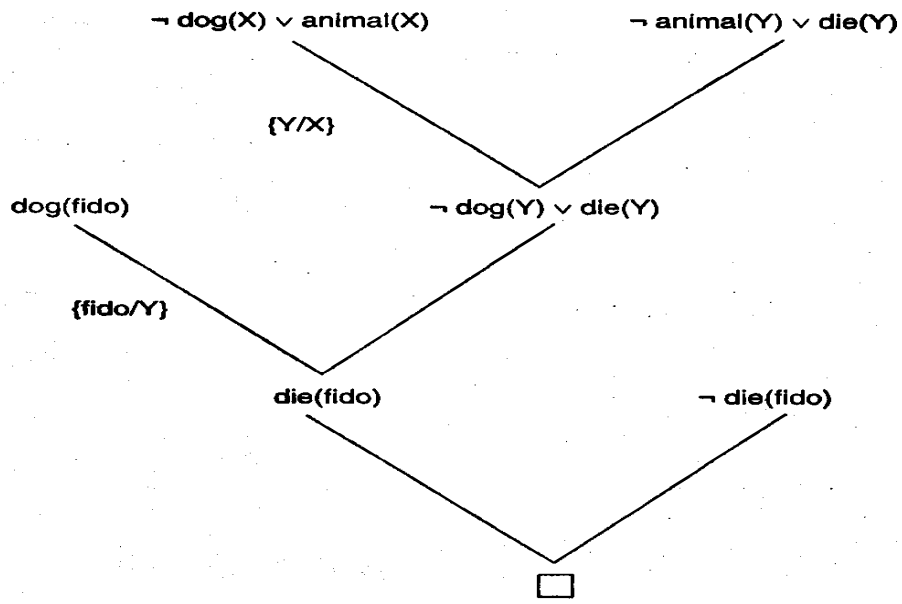


Figure (3-2): Resolution proof for the "dead dog" problem.

Ex(3): We now present another example of a resolution refutation for the predicate calculus. Consider the following story of the "lucky student":

"Anyone passing his history exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. John did not study but he is lucky. Anyone who is lucky wins the lottery. Is John happy?"

a. First change the sentences to predicate form:

1- Anyone passing his history exams and winning the lottery is happy.
$\forall X$ (pass(X, history) $\wedge$ win(X, lottery) $\Rightarrow$ happy(X))

2- Anyone who studies or is lucky can pass all his exams.
$\forall X \forall Y$ (study(X) $\vee$ lucky(X) $\Rightarrow$ pass(X,Y))

3- John did not study but he is lucky.
$\neg$ study(john) $\wedge$ lucky(john)

4- Anyone who is lucky wins the lottery.
$\forall X$ (lucky(X) $\Rightarrow$ win(X, lottery))

b. These four predicate statements are now changed to clause form:

1- $\neg$ pass(X, history) $\vee \neg$ win(X, lottery) $\vee$ happy(X)

2- $\neg$ study(Y) $\vee$ pass(Y,Z)

3- $\neg$ lucky(W) $\vee$ pass(W,V)

4- $\neg$ study(john)

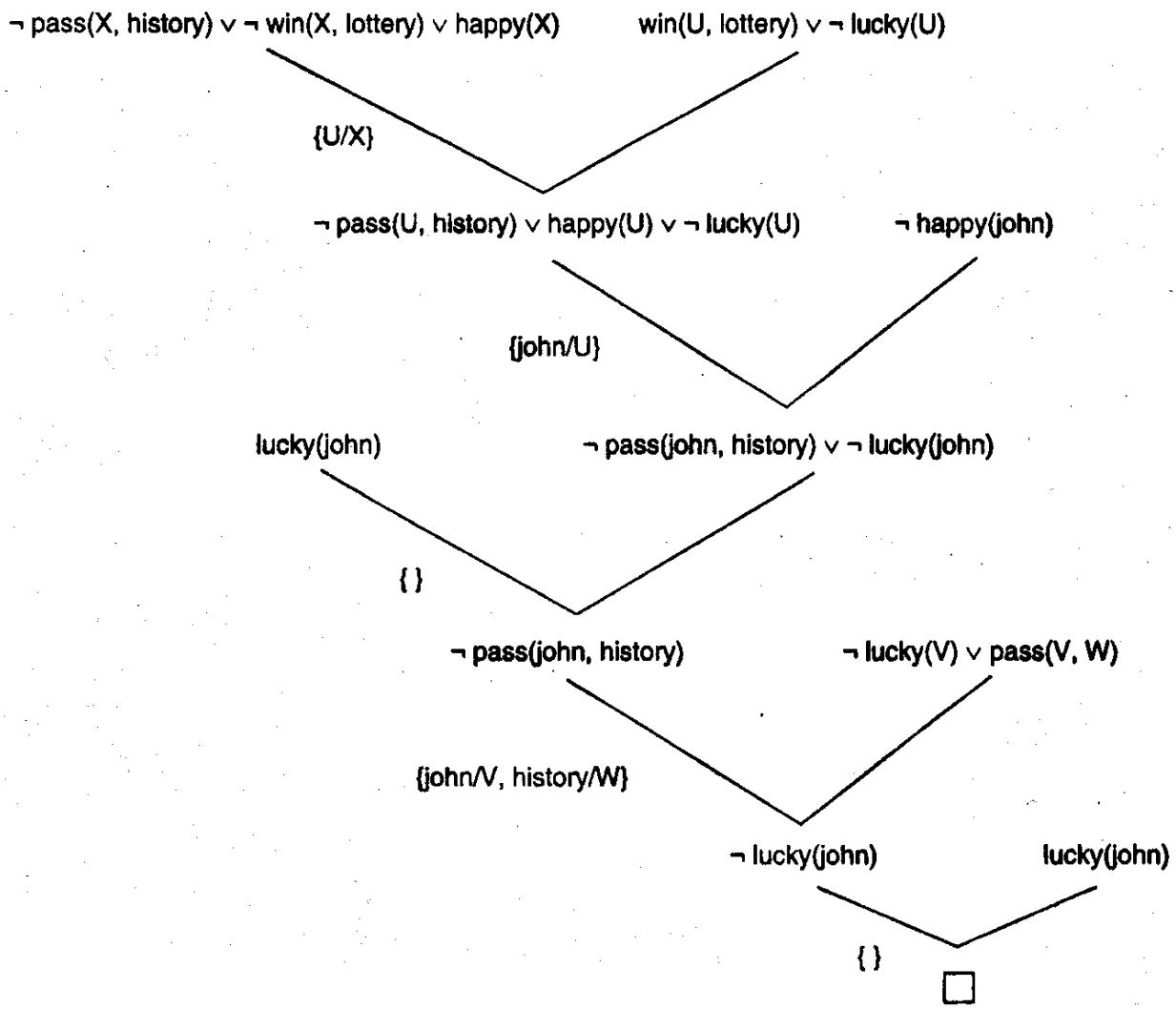
5- lucky(john)

6- $\neg$ lucky(U) $\vee$ win(U, lottery)

Into these clauses is entered, in clause form, the negation of the conclusion:

7- $\neg$ happy(john)

The resolution refutation graph of Figure (3-3) shows a derivation of the contradiction and, consequently, proves that John is happy.



Figure(3-3): One resultion refutation for the "happy student" problem.

Ex(4): As a final example, suppose:

"All people who are not poor and are smart are happy. Those people who read are not stupid. John can read and is wealthy. Happy people have exciting lives. Can anyone be found with an exciting life?"

a) First change the sentences to predicate form:

We assume ∀X (smart (X) ≡ ¬ stupid (X)) and ∀Y (wealthy (Y) ≡ ¬ poor (Y)), and get:

∀X (¬ poor (X) Λ smart (X) ⇒ happy (X))

∀Y (read (Y) ⇒ smart (Y))

read (john) Λ ¬ poor (john)
∀Z (happy (Z) ⇒ exciting (Z))
The negation of the conclusion is:

¬∃W (exciting (W))

b) These predicate calculus expressions for the happy life problem are transformed into the following clauses:

poor (X) v ¬ smart (X) v happy (X)

¬ read (Y) v smart (Y)

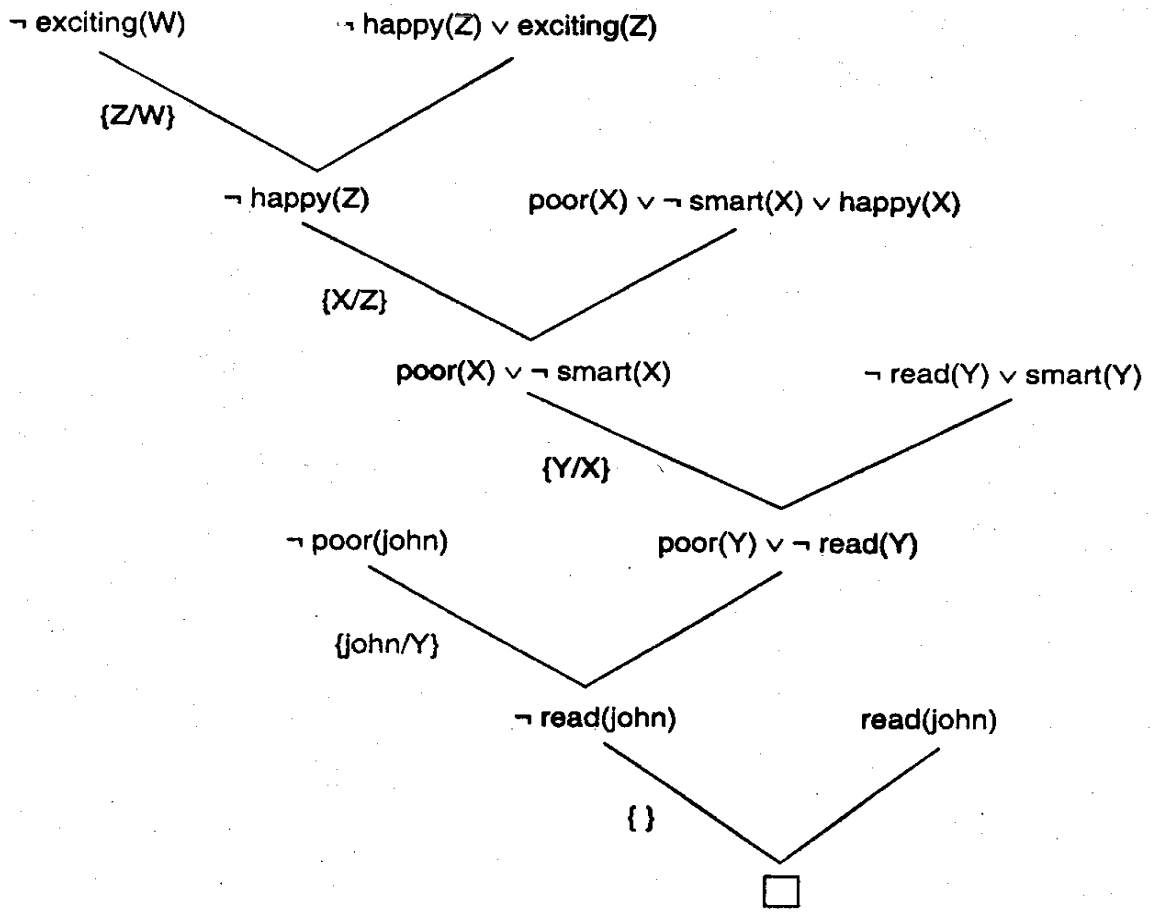
read (john)

¬ poor (john)

¬ happy (Z) v exciting (Z)

¬ exciting (W)

The resolution refutation for this example is found in Figure (3-4).



Figure(3-4): Resolution prove for the "exciting life" problem.

3.4 Answer Extraction from Resolution

Notice that inmost of our problems, the queries of the user are basically answered with either a Yes or a No. For instance, in the Exciting Life Problem, we are to find out if some person lives an exciting life or not. In our Fido example, we try to find out if Fido will die or not.

Resolution is not only capable of answering Yes or No queries, it is also able to yield specific answers to queries requesting for specific information. Consider the simple example below.

Fido the dog goes wherever John, his master, goes. John is at the library. Where is Fido?

First we represent this story in predicate calculus expressions and then reduce these expressions to clause form. The predicates:

at (john,X) → at (fido,X)

at (john,library)

The clauses:

¬ at (john,Y) ∨ at (fido,Y)

at (john,library)

The conclusion negated:

¬ at (fido,Z), Fido is nowhere!

Figure 3-5 gives the answer extraction process. The literal keeping track of unifications is the original question (where is Fido?):

at (fido,Z)

Once again, the unifications under which the contradiction is found tell how the original query is true: Fido is at the library.

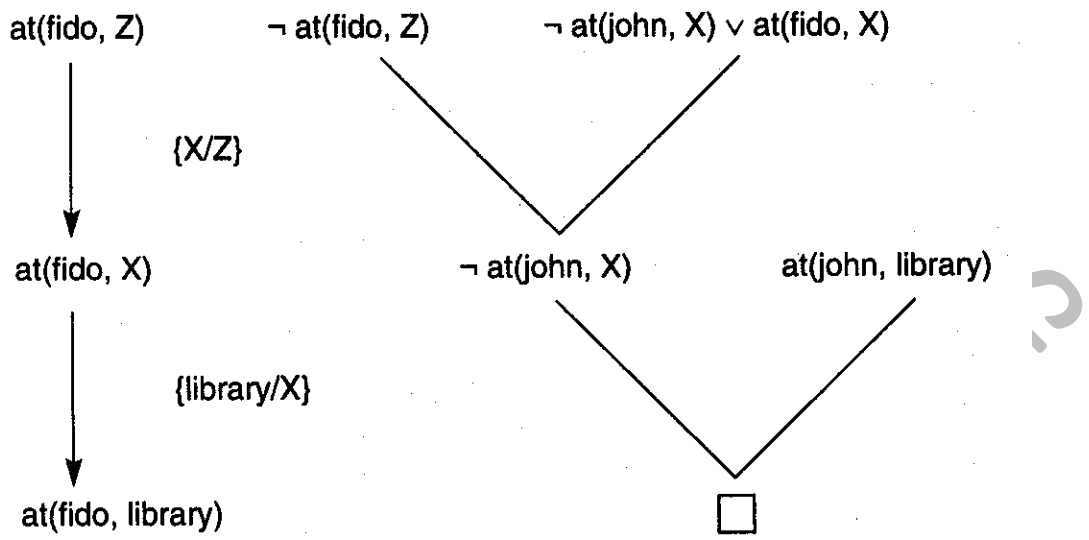


Figure 3-5 Answer extraction process on the "finding fido" problem.

The final example shows how the skolemization process can give the instance under which the answer may be extracted. Consider the following situation:

> Everyone has a parent. The parent of a parent is a grandparent. Given the person John, prove that John has a grandparent.

The following sentences represent the facts and relationships in the situation above. First, Everyone has a parent:

$$(\forall X)(\exists Y)\ p(X,Y)$$

A parent of a parent is a grandparent.

$$(\forall X)(\forall Y)(\forall Z)\ p(X,Y) \wedge p(Y,Z) \rightarrow gp(X,Z)$$

The goal is to find a W such that gp(john,W) or $\exists$ (W)(gp(john,W)). The negation of the goal is $\neg \exists$ (W)(gp(john,W)) or:

$$\neg\ gp(john,W)$$

In the process of putting the predicates above in clause form for the resolution refutation, the existential quantifier in the first predicate (everyone has a parent) requires a skolem function. This skolem function would be the obvious function: take the given X and find the parent of X. Let's call this the pa(X) for "find a parental ancestor for X ." For John this would be either his father or his mother. The clause form for the predicates of this problem is:

p(X,pa(X))

¬ p(W,Y) ∨ ¬ p(Y,Z) ∨ gp(W,Z)

¬ gp(john,V)

The resolution refutation and answer extraction process for this problem are presented in Figure 3-6 Note that the unification substitutions in the answer are

gp(john,pa(pa(john)))

The answer to the question of whether John has a grandparent is to "find the parental ancestor of John's parental ancestor." The skolemized function allows us to compute this result.

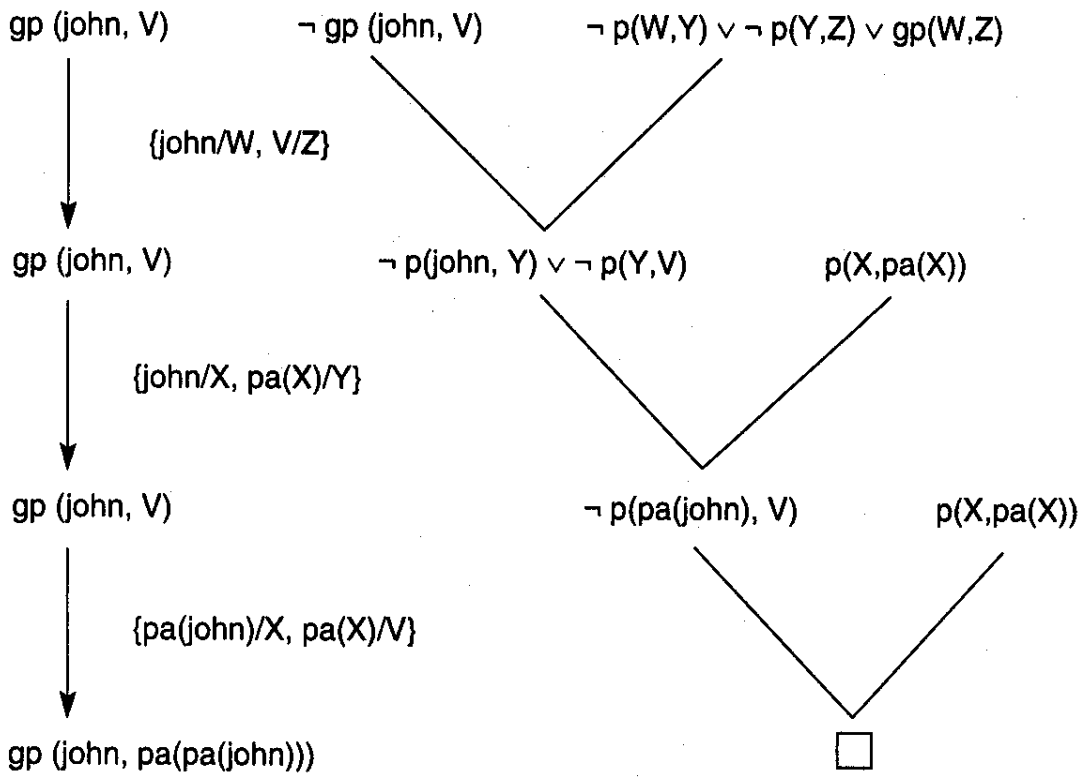


Figure 3-6 Skolemization as part of the answer extraction process.

Exercises

Q1: Write down the steps for converting an arbitrary predicate logic sentence into clausal form, and apply them to the following sentence:

$\forall X \forall Y \; p(X, Y) \Rightarrow \exists Z \; q(X, Z) \wedge \neg r(Y, Z)$

Q2: Convert the following predicate expression into clause form:

$\neg\{(\forall X)\ \{p(X) \Rightarrow \{(\forall Y)\ [p(Y) \Rightarrow p(f(X, Y))] \wedge \neg(\forall Y)\ [q(X, Y) \Rightarrow p(Y)]\}\}\}$

Q3: Skolemize $\forall X$ [$\exists Y$ animal (Y) ^ ¬ loves (X, Y)]v [$\exists Z$ loves(Z, X)].

Q4: consider the following: " Marcus is a man. Marcus is a pompeian. All pompeian are Romans. Caesar is a ruler. All Romans are either loyal to Caesar or hate him. Everyone is loyal to someone. People only try to assassinate rulers they are not loyal to. Marcus tried to assassinate Caesar." Prove using resolution that "Marcus hates Caesar".

Q5: Consider the following: "Ahmed likes all kinds of games. Football is game. Basketball is game. Anything anyone plays and isn't killed by is game. Ali plays tennis and still alive. Ban plays everything Ali plays." Use resolution to answer the question, "What game does Ban play?".