Resolution theorem proving

Resolution is a technique for proving theorems in the propositional or predicate calculus that has been a part of AI problem-solving research from the mid-1960s.

Resolution is the way of finding contradictions in a database of clauses with minimum use of substitution. Resolution refutation proves a theorem by negating the statement to be proved and adding this negated goal to the set of axioms that are known (have been assumed) to be true. It then uses the resolution rule of inference to show that this leads to a contradiction. Once the theorem prover shows that the negated goal is inconsistent with the given set of axioms, it follows that the original goal must be consistent. This proves the theorem.

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 (what was originally to be proven) 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".

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 ∨) 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. These are not intended to be proofs of the equivalence of these transformations across all predicate calculus expressions.

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 [ ], and remove redundant brackets. As an example, consider the following expression, where X, Y, and Z are variables and l a constant:

(i) $(\forall X)([a(X) \wedge b(X)] \rightarrow [c(X,l) \wedge (\exists Y)((\exists Z)[c(Y,Z)] \rightarrow d(X,Y))]) \vee (\forall X)(e(X))$

1. First we eliminate the → by using the equivalent form proved in Chapter 2: a → b ≡ ¬ a ∨ b. This transformation reduces the expression in (i) above:

(ii) $(\forall X)(\neg [a(X) \wedge b(X)] \vee [c(X,l) \wedge (\exists Y)((\exists Z)[\neg c(Y,Z)] \vee d(X,Y))] \vee (\forall X)(e(X))$

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

$\neg (\neg a) \equiv a$

$\neg (\exists X)\, a(X) \equiv (\forall X) \neg a(X)$

$\neg (\forall X)\, b(X) \equiv (\exists X) \neg b(X)$

$\neg (a \wedge b) \equiv \neg a \vee \neg b$

$\neg (a \vee b) \equiv \neg a \wedge \neg b$

Using the fourth equivalences (ii) becomes:

(iii) $(\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X,l) \wedge (\exists Y)((\exists Z)[\neg c(Y,Z)] \vee d(X,Y))]) \vee (\forall X)(e(X))$

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:

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

(iv) $(\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\exists Y)((\exists Z)[\neg c(Y,Z)] \vee d(X,Y))]) \vee (\forall W)(e(W))$

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) $(\forall X)(\exists Y)(\exists Z)(\forall W)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\neg c(Y,Z) \vee d(X,Y))] \vee e(W))$

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:

$(\exists X)(dog(X))$ 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:

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

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:

$(\forall X)\ mother(X, m(X))$

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

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

is skolemized to:

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

The existentially quantified Y and Z are within the scope universally quantified X but not within the scope of W. Thus each will be replaced by a skolem function of X. Replacing Y with the skolem function f(X) and Z with g(X), (v) becomes:

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

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

7. Next we convert the expression to the conjunct of disjuncts form. This requires using the associative and distributive properties of $\wedge$ and $\vee$. Recall

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

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

which indicates that $\wedge$ or $\vee$ 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, $\wedge$ is not distributed. However, $\vee$ must be distributed across $\wedge$ using:

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

The final form of (vii) is:

(viii) $[\neg a(X) \vee \neg b(X) \vee c(X,l) \vee e(W)] \wedge$
$[\neg a(X) \vee \neg b(X) \vee \neg c(f(X),g(X)) \vee d(X,f(X)) \vee e(W)]$

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

(ixa) $\neg a(X) \vee \neg b(X) \vee c(X,l) \vee e(W)$

(ixb) $\neg a(X) \vee \neg b(X) \vee \neg c(f(X),g(X)) \vee d(X,f(X)) \vee e(W)$

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.

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

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) ∨ ¬ b(X) ∨ c(X,l) ∨ e (W)

(xb) ¬ a(U) ∨ ¬ b(U) ∨ ¬ c(f(U),g(U)) ∨ d (U,f(U)) ∨ e (V)

Binary resolution proof procedure

The *resolution refutation* proof procedure answers a query or deduces a new result by reducing a 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.

Example: We wish to prove that "**Fido will die**" from the statements that "**Fido is a dog**" and "**all dogs are animals**" and "**all animals will die**."

Changing these three premises to predicates gives:

Fido is a dog: dog (fido).

All dogs are animals: ∀(X) (dog (X) → animal (X)).

All animals will die: ∀(Y) (animal (Y) → die (Y)).

converts these predicates to clause form:

PREDICATE FORM	CLAUSE FORM
dog (fido)	dog (fido)
∀(X) (dog) (X) → animal (X))	¯ dog (X) ∨ animal (X)
∀(Y) (animal (Y) → die (Y))	¯ animal (Y) ∨ die (Y)

Negate the conclusion that Fido will die:

¯ die (fido) ¯ die (fido)

Resolve clauses having opposite literals, producing new clauses by resolution as in Figure .This process is often called *clashing*.

The symbol □ in Figure below indicates that the empty clause is produced and the contradiction found. The □ symbolizes the clashing of a predicate and its negation: the situation where two mutually contradictory statements are present in the clause space. These are clashed to produce the empty clause. The sequence of substitutions (unifications) used to make predicates equivalent also gives us the value of variables under which a goal is true.

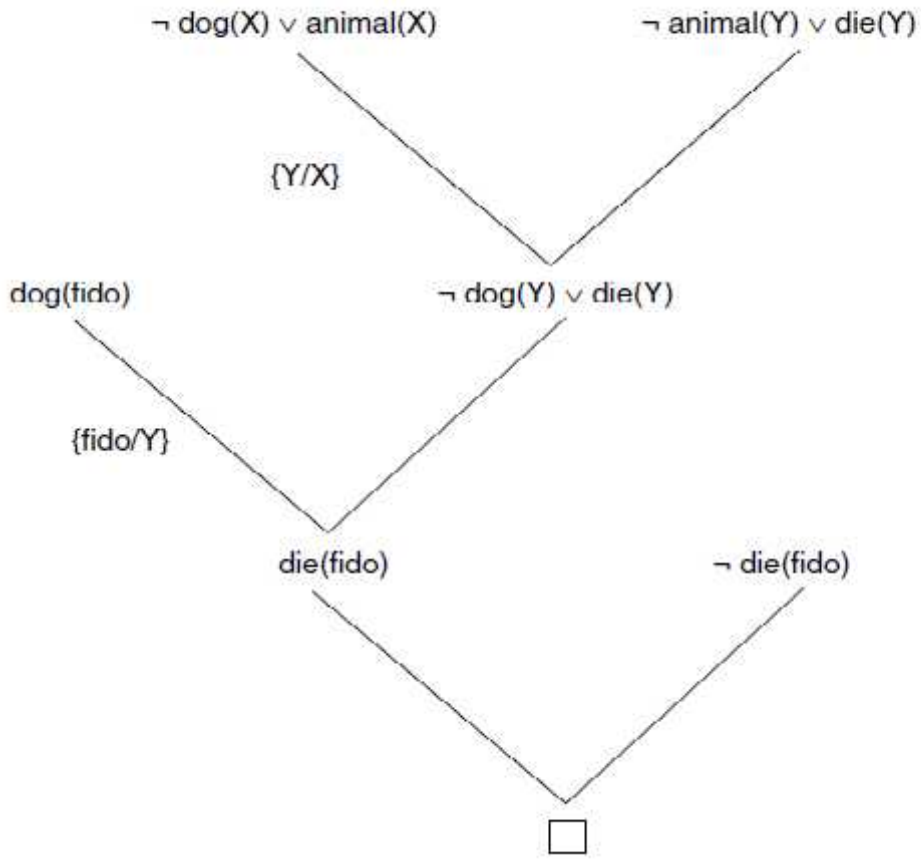

Figure 1: Resolution proof for the ***"dead dog"*** problem.

Example 2: We now present an example of a resolution refutation for the predicate calculus. Consider the following story of the "*happy 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?**

- ***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))

• ***Second changed to clause form***

1. ¬ pass (X, history) ∨ ¬ win (X, lottery) ∨ happy (X)
2. ¬ study (Y) ∨ pass (Y, Z)
3. ¬ lucky (W) ∨ pass (W, V)
4. ¬ study (john)
5. lucky (john)
6. ¬ lucky (U) ∨ win (U, lottery)

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

7. ¬ happy (john)

The resolution refutation graph of Figure (2) shows a derivation of the contradiction and, consequently, proves that John is happy.

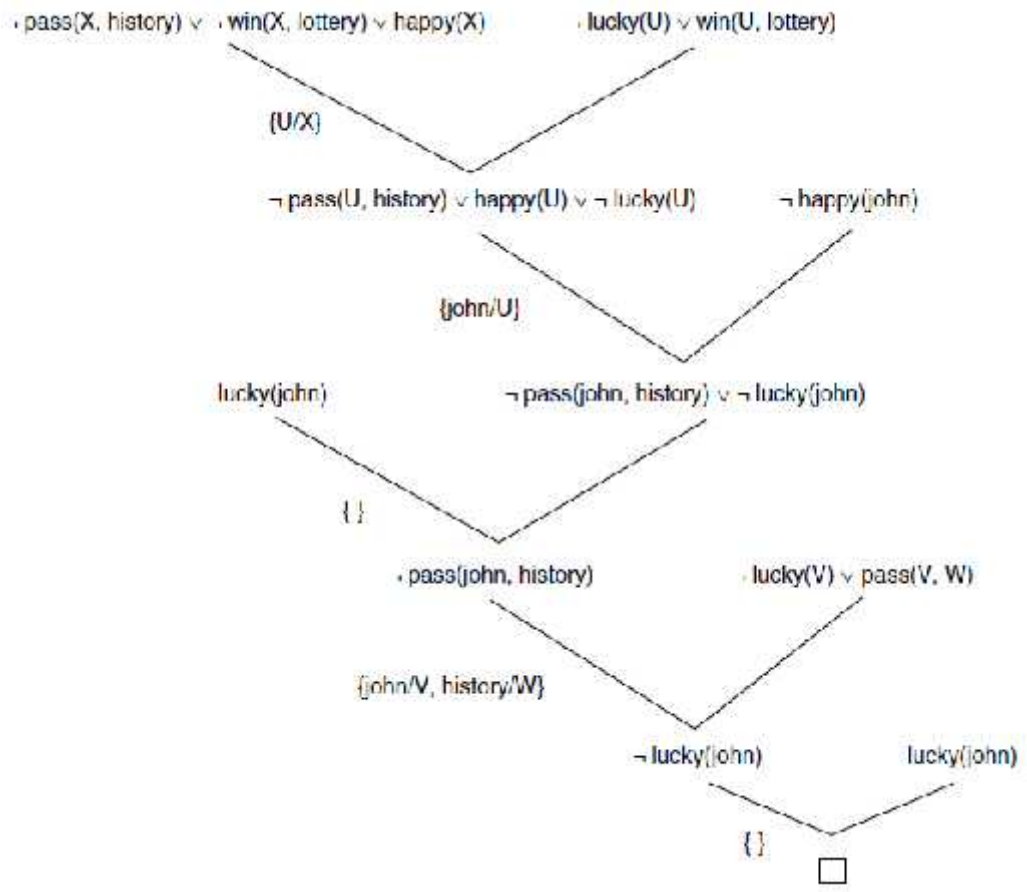

Figure 2: resolution refutation for the ***happy student*** problem.

Example3: As a final example in this subsection we present the "exciting life" problem; 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?

• ***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) ∧ smart (X) → happy (X))

∀Y (read (Y) → smart (Y))

read (john) ∧ ¬ poor (john)

∀Z (happy (Z) → exciting (Z))

The negation of the conclusion is:

¬ ∃ W (exciting (W))

• ***Second changed to clause form***

poor (X) ∨ ¬ smart (X) ∨ happy (X)

¬ read (Y) ∨ smart (Y)

read (john)

¬ poor (john)

¬ happy (Z) ∨ exciting (Z)

¬ exciting (W)

The resolution refutation for this example is found below Figure (3)

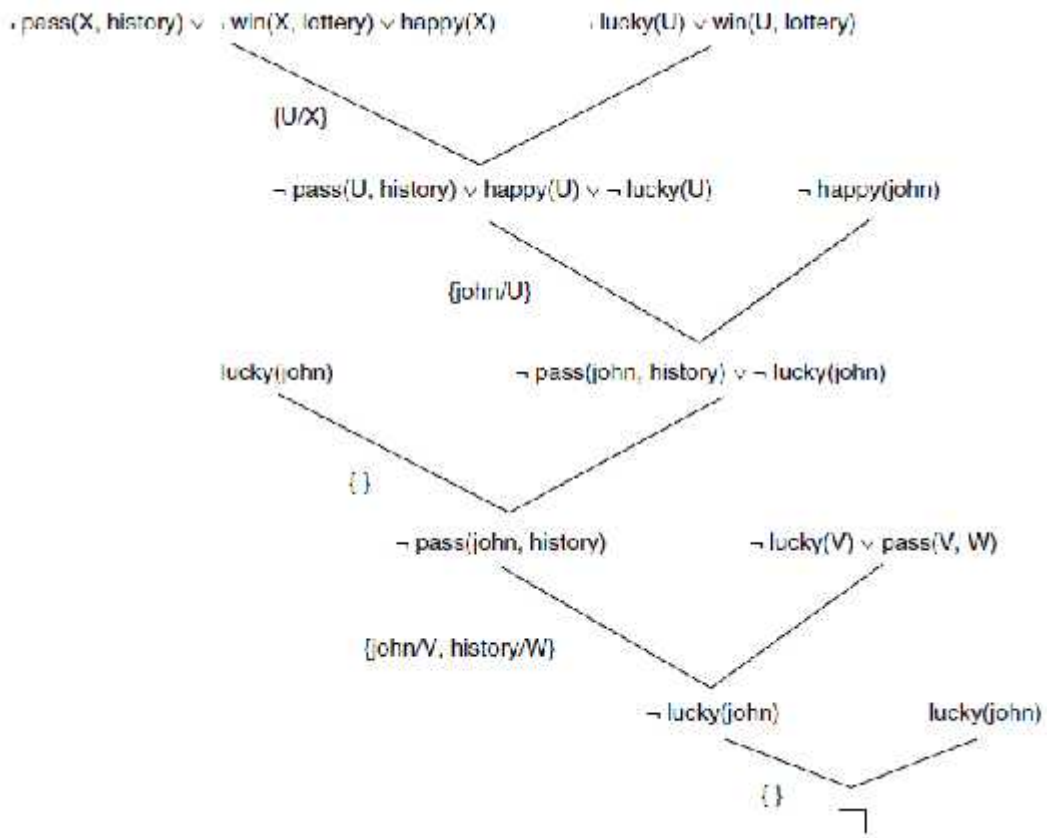

Figure 3: Resolution prove for *"**exciting life**"* problem