ME 620: Fundamentals of Artificial Intelligence

Lecture 18: Inference in FOL - Part II

Shyamanta M Hazarika
Biomimetic Robotics and Artificial Intelligence Lab
Mechanical Engineering and M F School of Data Sc. & AI
IIT Guwahati

Resolution

- **Resolution is an important rule of inference** that can be applied to a certain class of well-formed formulas called clauses.
  - First step in applying resolution is to convert **FOPC well-formed formula to clausal normal form**.
  - A clause is a set of literals representing their disjunction.
    - A literal is an atomic sentence or the negation of an atomic sentence.
- The resolution process, when applicable, is applied to a pair of parent clauses to **produce a derived clause**.

Resolution for Ground Clauses

- We have two ground clauses:
  1. $P_1 \vee P_2 \vee P_3 \ldots . \vee P_N$

     No terms contain variables; we have a ground instance of the literal.
  2. $\neg P_1 \vee Q_1 \vee Q_2 \vee Q_3 \ldots .. \vee Q_M$

  - Assume in above **ground clauses** all of P_i and Q_j are distinct.
  - One of these clauses contains a literal that is the exact negation of one of the literals in the other clause.

    From the two above parent clauses; infer a new clause, called the resolvent of the two.
  - The **resolvent** is computed by taking the **disjunction of the two clauses** after **eliminating the complementary pair** P1 and ¬ P1.

- Resolution allows for **incorporation of several operations** into **one simple inference rule**.

Clauses and Resolvents

Parent Clause	Resolvents	Comments
1. P 2. $\neg P \vee Q$	Q	Modus Ponens
1. $P \vee Q$ 2. $\neg P \vee Q$	Q	Merge
1. $P \vee Q$ 2. $\neg P \vee \neg Q$	$Q \vee \neg Q$ $P \vee \neg P$	Tautologies
1. P 2. $\neg P$	$\square$	Empty Clause
1. $\neg P \vee Q$ 2. $\neg Q \vee R$	$\neg P \vee R$	Chaining

General Resolution Principle

Definition: Suppose $\mathbf{\Phi}$ and $\mathbf{\Psi}$ are two clauses. If there is a literal ϕ in $\mathbf{\Phi}$ a literal $\neg\psi$ in $\mathbf{\Psi}$ such that ϕ and ψ have a most general unifier γ, then we can **infer the clause** obtained by **applying the substitution** γ to the **union of $\mathbf{\Phi}$ and $\mathbf{\Psi}$ minus the complementary literals**.

$$\frac{\begin{array}{l}\mathbf{\Phi} \\ \mathbf{\Psi}\end{array}}{((\mathbf{\Phi} - \{\phi\}) \cup (\mathbf{\Psi} - \{\neg\psi\}))\,\gamma} \qquad \begin{array}{l}\text{with } \phi \in \mathbf{\Phi} \\ \text{with } \neg\psi \in \mathbf{\Psi} \\ \text{where } \phi\gamma = \psi\gamma\end{array}$$

Resolution Derivation

<u>Definition</u>: The **resolution derivation** of a clause Φ from a set of clauses Δ is a **sequence of clauses** in which

a. Clause Φ **is the last element of the sequence**, and

b. Each **element is either a member of** Δ or **result of applying the resolution** principle to clauses earlier in the sequence.

We write

$$\Delta \vdash \Phi$$

if there **exits a derivation of Φ from Δ**.

Resolution Derivation

$I(x)$: `x' is intelligent

$H(x)$: `x' has commonsense

D : Deep Blue

<u>Example</u>

1.	$\neg I(x) \vee H(x)$	Δ
2.	$\neg H(D)$	Δ
3.	$I(D)$	Δ
4.	$\neg I(D)$	1,2
5.	$\square$	3,4

Clause in line 4 is derived from clauses in line 1 and 2.

The empty clause is derived from clauses in line 3 and 4.

We can write $\Delta \vdash \square$

The following sequence of clauses is a resolution derivation of the empty clause from the set of clauses labelled Δ

Resolution Graph

1. P Δ
2. ¬ P ∨ Q Δ
3. ¬ Q ∨ R Δ
4. ¬ R Δ

Three-Level Resolution Graph

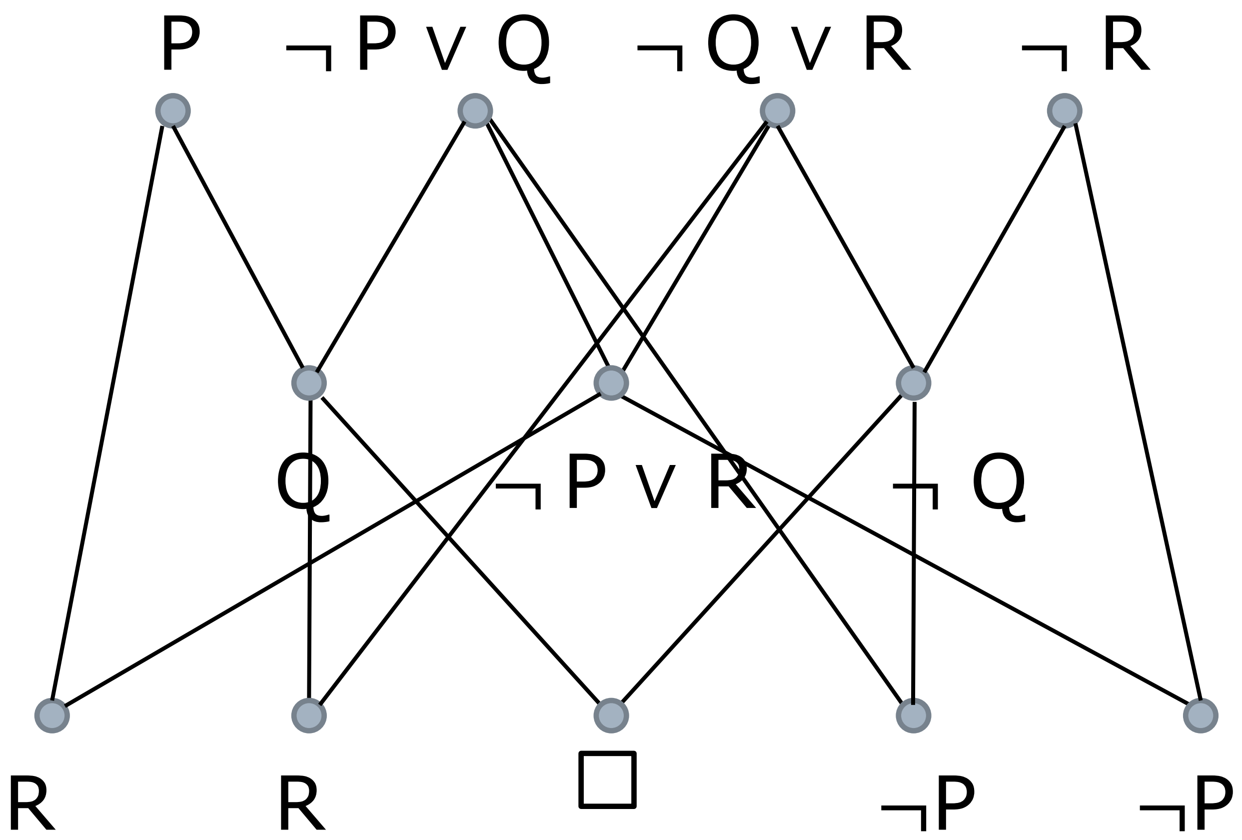


Figure on the right shows the Resolution graph: graph of possible resolutions from the initial database.

Expanded to three levels of deduction.

We want to encode such graphs in a linear form.

One of the problems of with such inference graphs is that they are difficult to be followed!

Resolution Trace

Definition: The **resolution trace** is a **sequence of annotated clauses separated into levels**.

a. The **first level** contains the **clauses in the initial database**.
b. Each subsequent level contains all clauses with at least one parent at the previous level.
c. The **annotations specify the clauses from which they are derived**.

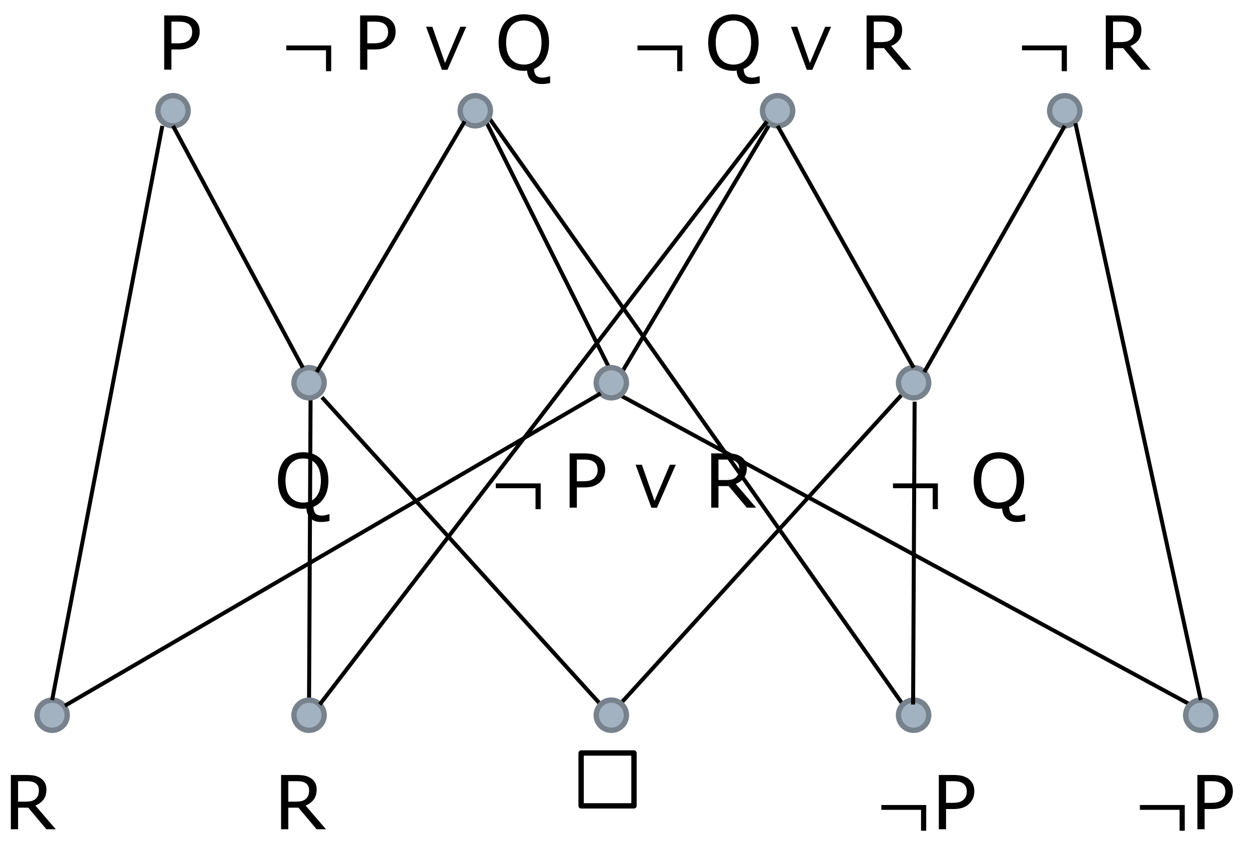
A resolution trace **captures the information of a resolution graph**, in a linear form.

Resolution Trace

1.	P	Δ
2.	$\neg P \vee Q$	Δ
3.	$\neg Q \vee R$	Δ
4.	$\neg R$	Δ
5.	Q	1,2
6.	$\neg P \vee R$	2,3
7.	$\neg Q$	3,4

A resolution trace captures the information of a resolution graph, in a linear form.

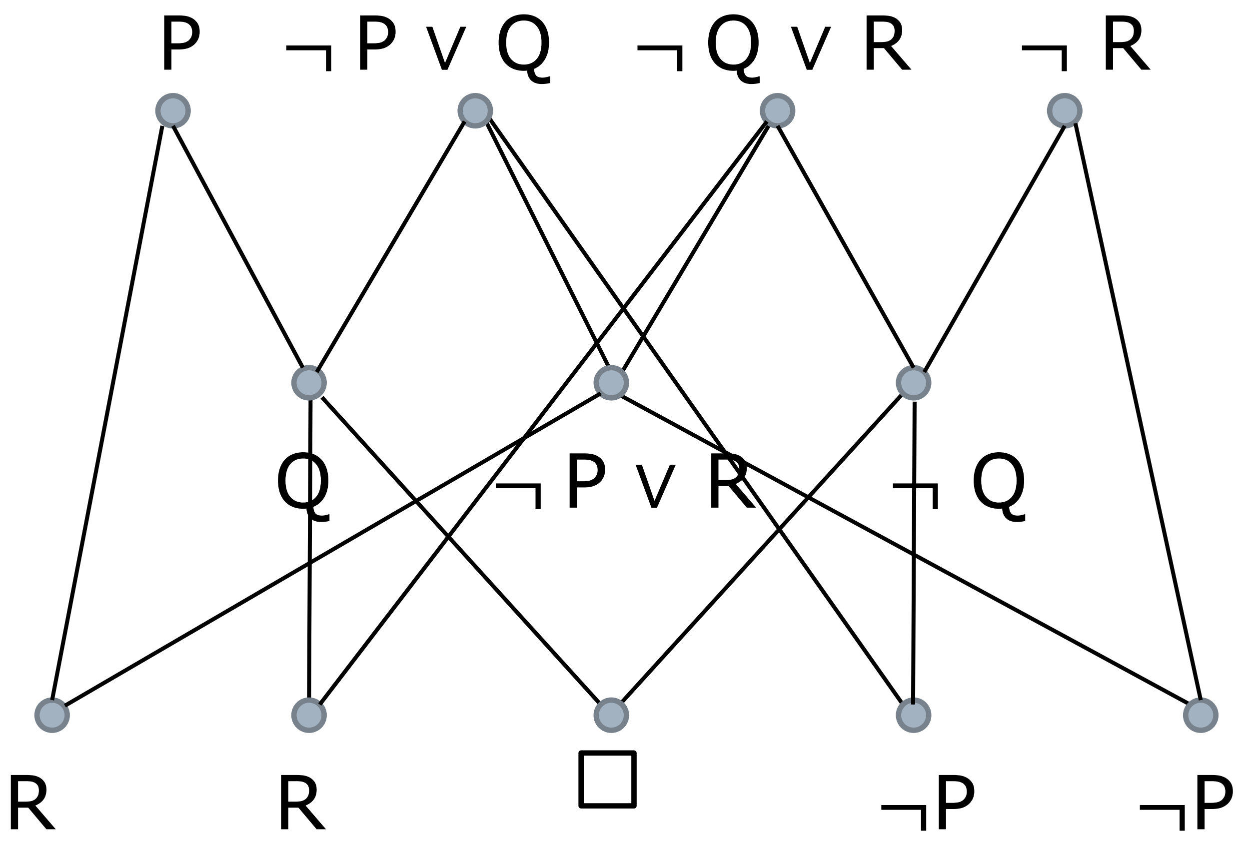
Three-Level Resolution Graph



Resolution Trace

1.	P	Δ
2.	¬P ∨ Q	Δ
3.	¬Q ∨ R	Δ
4.	¬R	Δ
5.	Q	1,2
6.	¬P ∨ R	2,3
7.	¬Q	3,4
8.	R	3,5
9.	R	1,6
10.	¬P	4,6
11.	¬P	2,7
12.	□	5,7

Three-Level Resolution Graph



The resolution trace searches the inference graph in a breadth-first fashion.

Resolution Refutation

Definition: The **resolution refutation** of a set Δ of clauses, is a sequence $\Phi_1, \Phi_2 \ldots \Phi_n$ of clauses such that

a. Every Φ_i is either in Δ or follows by resolution (i.e., it is a resolvent) of prior clauses in the sequence.

b. In addition, **we require that** $\Phi_\mathrm{n} = \square$.

If there is a resolution refutation of Δ, then we say that Δ **is refutable**.

Resolution Refutation

1.	P	Δ
2.	$\neg P \vee Q$	Δ
3.	$\neg Q \vee R$	Δ
4.	$\neg R$	Δ
5.	Q	1,2
6.	$\neg P \vee R$	2,3
7.	$\neg Q$	3,4
8.	R	3,5
9.	R	1,6
10.	$\neg P$	4,6
11.	$\neg P$	2,7
12.	$\square$	5,7

Sequence Φ_1, Φ_2 … Φ_n of clauses with

Φ_n = [] .

Resolution refutation of Δ.

Set of clauses is unsatisfiable.

Unsatisfiability

- Resolution is used in demonstrating **unsatisfiability**.
  - If a set of clauses is unsatisfiable, it is **always possible by resolution to derive a contradiction** from clauses in the set.

    In clausal form, a contradiction takes the form of an empty clause, which is equivalent to a disjunction of no literals.

  - Demonstrating **unsatisfiability for a set of clauses** can also be used to **demonstrate that a formula is logically implied** by a set of formula.
    - For a set of clauses Δ, we can show that Δ logically implies a formula Φ by finding a proof of Φ from Δ i.e., by establishing $\Delta \vdash \Phi$
    - By the **refutation theorem**, we can establish $\Delta \vdash \Phi$ by showing that $\Delta \cup \{\neg\Phi\}$ is inconsistent (unsatisfiable).
  - If we show that the set of formula $\Delta \cup \{\neg\Phi\}$ is unsatisfiable, we have demonstrated that Δ logically implies Φ

Unsatisfiability

From the standpoint of Models

If $\Delta \models \Phi$

The domain paired with an interpretation is a *model* for the language. A model for a first-order language is directly analogous to a truth assignment for propositional logic, because it provides all the information we need to determine the truth value of each sentence in the language.

Then **all models of Δ are also models of Φ**. None of these can be models of $\neg\Phi$, and thus $\Delta \cup \{\neg\Phi\}$ is unsatisfiable.

Conversely, $\Delta \cup \{\neg\Phi\}$ is unsatisfiable; Δ is satisfiable.

Let I be an interpretation that satisfies Δ; **I does not satisfy** $\neg\Phi$; for if it did, $\Delta \cup \{\neg\Phi\}$ would be satisfiable. **Therefore I satisfies** Φ.

Since this holds for arbitrary I satisfying Δ, it holds for all I satisfying Δ. Thus **all models of Δ are also models of Φ**, and Δ logically implies Φ.

Resolution Refutation Systems

- To turn this into a **proof technique**, we take a **set of clauses Δ together with negation of the goal** $\{\neg\Phi\}$; and try to show that it is inconsistent and hence **lead to a contradiction**.
  - If there is no contradiction then the clauses must be satisfiable.
  - If there is a contradiction then the $\Delta \cup \{\neg\Phi\}$ must be unsatisfiable. The **process of finding a contradiction is called finding a refutation**.
  - If empty clause is produced from the clause set $\Delta \cup \{\neg\Phi\}$ then $\Delta \vdash \Phi$

Resolution Refutation Systems

- In a **resolution refutation**, we first negate the goal wff, Φ. Negated goal is added to the set of wffs, Δ, from which we wish to prove Φ.

$$\Delta \cup \{\neg\Phi\}$$

- If Φ logically follows from Δ, the set $\Delta \cup \neg \Phi$ is *unsatisfiable*. Applying **resolution repeatedly to a set of unsatisfiable clauses**, eventually produces the empty clause.

$$\Delta \cup \{\neg\Phi\} \vDash \square$$

  - If Φ logically follows from Δ, then resolution eventually will produce $\square$ from the clause representation of $\Delta \cup \{\neg\Phi\}$
  - Conversely, if the empty clause $\square$ is produced from the clause representation of $\Delta \cup \{\neg\Phi\}$, then Φ logically follows from Δ.

Resolution Refutation Proof

Basic steps for proving a conclusion Φ given premises

Premise_1 , …, Premise_n

All of the premises expressed in FOL.

1. Convert all sentences to **Clausal Normal Form** (CNF)
2. Negate conclusion Φ and convert result to CNF
3. Add negated conclusion $\neg\Phi$ to the premise clauses.
4. Repeat until **contradiction** or no progress is made:
   a. Select 2 clauses (call them parent clauses)
   b. Resolve them together, performing all required unifications
   c. If resolvent is the empty clause, a contradiction has been found (i.e., Φ follows from the premises)
   d. If not, add resolvent to the premises.

If we succeed in Step 4, we have proved the conclusion.

Soundness and Completeness

Resolution is Sound

- Resolution is sound in that **any clause that can be derived** from a database using resolution is **logically implied** by that database.

  Soundness Theorem: If there is a resolution derivation of a clause Φ from a set of clauses Δ, then Δ logically implies Φ.

  Special case of this theorem; the check for unsatisfiability.

  - If there is a **deduction of the empty clause** from a database , then the database must logically imply the empty clause and, therefore, is **unsatisfiable**.

Soundness and Completeness

Resolution is not Complete

- Resolution is not complete! By itself, **it would not generate every clause that is logically implied** by a given set of clauses.

Tautology – something that is always true: [P ∨ ¬P].

  - For example, Tautology is logically implied by every set of clauses but resolution will not produce this clause from the empty clause.
  - It provides **no way of using sentences involving the equality and inequality relations**.
    - For example, Given a database consisting of sentences P(A) and A=B; resolution cannot prove P(B). This is because as far as the database, the relation constant = is arbitrary.

  To give its standard interpretation, additional axioms are required.

Soundness and Completeness

Resolution is Refutation Complete

- For a database **without sentences involving the equality and inequality relations**, the procedure is refutation complete, i.e.,

  Given an unsatisfiable set of sentences, it is guaranteed to produce the empty clause.

  - Consequently, we can **turn it into a proof technique** to **determine logical implication** by negating the clause to be proved, adding it to the database, and proving its unsatisfiability.

The proof of refutation completeness is a little complicated and involves the introduction of several new concepts and lemmas. It is outside the scope of this discussion on Introduction to KR & R.

Resolution and Equality

- **Refutation completeness** of resolution **does not hold for databases containing** the **relation constant =** intended to be interpreted as equality relation.
  - No mechanism for substitution of nonvariable terms that are known to be equal.
- One way of dealing with sentences involving equality is to **axiomatize the equality relation**
  - Reflexive $\forall x \; x = x$
  - Symmetric $\forall x \forall y \; [x = y \rightarrow y = x]$
  - Transitive $\forall x \forall y \forall z \; [x = y \wedge y = z \rightarrow x = z]$
- **Supply appropriate substitution axioms**
  - Substitute terms for terms in each of functions and relations.

Resolution and Equality

- Of course, this would work if we have **substitution axiom for every function and relation** within which we want substitutions to occur.
  - Tedious to write these axioms in situations involving numerous functions and relations.
- Rule of inference, called **paramodulation** when added to resolution principle, **guarantees refutation completeness**, even **when involving equality**.
  - Weaker version of paramodulation, called demodulation which is easier to understand and efficient.
  - **Demodulation is the basis for the semantics of functional programming languages such as LISP**.

Resolution Refutation Proofs

Example 1

Premise

1. If a course is easy, some students are happy.
2. If a course has a final exam, no students are happy.

Prove: If a course has a final exam, the course is not easy.

Recall that a predicate is an assertion that some property or relationship holds for one or more arguments,

Predicates

easy(x) : Course \`x' is a easy.

happy(x) : Student \`x' is happy.

final(x) : Course \`x' has a final exam.

Resolution Refutation Proofs

1. **If a course is easy, some students are happy.**

$\forall x\,[\mathrm{easy}(x) \rightarrow \exists y\,\mathrm{happy}(y)]$

$\forall x\,[\neg\mathrm{easy}(x) \vee \exists y\,\mathrm{happy}(y)]$

$\forall x\,[\neg\mathrm{easy}(x) \vee \mathrm{happy}(f(x))]$

$[\neg\mathrm{easy}(x) \vee \mathrm{happy}(f(x))]$

C1. $[\neg\mathrm{easy}(x_1) \vee \mathrm{happy}(f(x_2))]$

2. **If a course has a final exam, no students are happy.**

$\forall x\,[\mathrm{final}(x) \rightarrow \neg\exists y\,\mathrm{happy}(y)]$

$\forall x\,[\mathrm{final}(x) \rightarrow \forall y\neg\mathrm{happy}(y)]$

$\forall x\forall y\,[\mathrm{final}(x) \rightarrow \neg\mathrm{happy}(y)]$

$[\neg\,\mathrm{final}(x) \vee \neg\mathrm{happy}(y)]$

C2. $[\neg\,\mathrm{final}(x_3) \vee \neg\mathrm{happy}(y_1)]$

Resolution Refutation Proofs

Φ: **If a course has a final exam, the course is not easy.**

$\forall x\,[\text{final}(x) \rightarrow \neg\exists y\, \text{easy}(y)]$

$\forall x\,[\text{final}(x) \rightarrow \forall y \neg \text{easy}(y)]$

$\forall x \forall y\,[\text{final}(x) \rightarrow \neg\, \text{easy}(y)]$

. $[\neg\, \text{final}(x_4) \vee \neg\, \text{easy}(y_2)]$

Negate the goal: $\neg\,[\neg\, \text{final}(x_4) \vee \neg\, \text{easy}(y_2)]$

$[\text{final}(x_4) \wedge \text{easy}(y_2)]$

C3. $\text{final}(x_4)$

C4. $\text{easy}(y_2)$

Resolution Refutation Proofs

Resolution Trace

1.	$[\neg easy(x_1) \lor happy(f(x_2))]$	C1
2.	$[\neg final(x_3) \lor \neg happy(y_1)]$	C2
3.	$final(x_4)$	C3
4.	$easy(y_2)$	C4
5.	$\neg happy(y_1)$	2,3
6.	$happy(f(x_2))$	1,4
7.	$\square$	5,6

Intuitively, 5 states that no student is happy; 6 states that a particular student $f(x_2)$ is happy.

We have arrived at a contradiction. AFTER the goal was negated and added to the clause set. Hence, the statement is proved.

Resolution Refutation Proofs

Example 2

Premise

1. The father of someone or the mother of someone is an ancestor of that person.
2. An ancestor of someone's ancestor is also the ancestor of that person.
3. Jesse is the father of David.
4. David is the ancestor of Mary
5. Mary is the mother of Jesus

Prove: Jesse is an ancestor of Jesus.

Resolution Refutation Proofs

Predicates

A predicate is an assertion that some property or relationship holds for one or more arguments,

1. father(x,y) : `x' is the father of `y'.
2. mother(x,y) : `x' is the mother of `y'.
3. ancestor(x,y) : `x' is the ancestor of `y'.

Constants

1. Jesse
2. David
3. Mary
4. Jesus

A constant is a symbolic name for a real-world person, object or event.

Resolution Refutation Proofs

1. The father of someone or the mother of someone is an ancestor of that person

$\forall x \forall y[(father(x,y) \vee mother(x,y)) \rightarrow ancestor(x,y)]$

$\forall x \forall y[\neg (father(x,y) \vee mother(x,y)) \vee ancestor(x,y)]$

$\forall x \forall y[(\neg father(x,y) \wedge \neg mother(x,y)) \vee ancestor(x,y)]$

$\forall x \forall y[(\neg father(x,y) \vee ancestor(x,y)) \wedge (\neg mother(x,y)) \vee ancestor(x,y)]$

C1. $[\neg father(x_1,y_1) \vee ancestor(x_1,y_1)]$

C2. $[\neg mother(x_2,y_2) \vee ancestor(x_2,y_2)]$

2. An ancestor of someone's ancestor is also an ancestor of that person.

$\forall r \forall s \forall t[(ancestor(r,s) \wedge ancestor(s,t)) \rightarrow ancestor(r,t)]$

$\forall r \forall s \forall t[\neg (ancestor(r,s) \wedge ancestor(s,t)) \vee ancestor(r,t)]$

$\forall r \forall s \forall t[\neg ancestor(r,s) \vee \neg ancestor(s,t) \vee ancestor(r,t)]$

C3. $[\neg ancestor(r,s) \vee \neg ancestor(s,t) \vee ancestor(r,t)]$

Resolution Refutation Proofs

3. Jesse is the father of David.

   C4. father(Jesse, David).

4. David is an ancestor of Mary.

   C5. ancestor(David, Mary).

5. Mary is the mother of Jesus.

   C6. mother(Mary, Jesus).

Φ: Jesse is an ancestor of Jesus.

ancestor(Jesse, Jesus).

Negate the goal: ¬ ancestor(Jesse, Jesus).

C7. ¬ ancestor(Jesse, Jesus).

Resolution Refutation Proofs

Resolution Trace

We have arrived at a contradiction; Hence, the statement is proved.

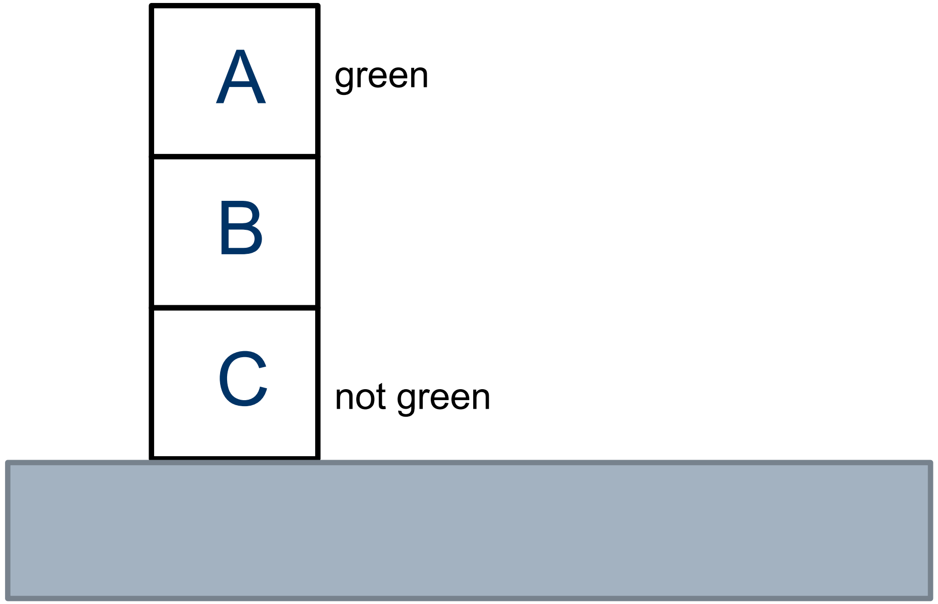
1.	$[\neg$ father$(x_1,y_1) \vee$ ancestor$(x_1,y_1)]$	C1	
2.	$[\neg$ mother$(x_2,y_2) \vee$ ancestor$(x_2,y_2)]$	C2	
3.	$[\neg$ ancestor(r,s) $\vee \neg$ ancestor(s,t) $\vee$ ancestor(r,t)$]$	C3	
4.	father(Jesse, David)	C4	
5.	ancestor(David, Mary)	C5	
6.	mother(Mary, Jesus)	C6	
7.	$\neg$ ancestor(Jesse, Jesus).	C7	
8.	$\neg$ ancestor(Jesse,s) $\vee \neg$ ancestor(s,Jesus)	3,7	{Jesse/r, Jesus/t}
9.	$\neg$ father(Jesse,s) $\vee \neg$ ancestor(s,Jesus)	1,8	{Jesse/x_1, s/y_1}
10.	$\neg$ ancestor(David,Jesus)	4,9	{David/s}
11.	$\neg$ ancestor(David,s) $\vee \neg$ ancestor(s,Jesus)	3,10	{David/r, Jesus/t}
12.	$\neg$ ancestor(Mary,Jesus)	5,11	{Mary/s}
13.	$\neg$ mother(Mary,Jesus)	2,12	{Mary/x_2, Jesus/y_2}
14.	$\square$	6,13	

Resolution Refutation Proofs

Example 3

Suppose there are three coloured blocks stacked as shown, where the top one is \`green' and the bottom is \`not green'. Is there a \`green' block on top of a \`non-green' block?

on: holds if and only if block is immediately above the other.



Predicates

on(x,y): holds if and only if block \`x' is immediately above \`y'

green(x): block \`x' is green.

Query

$$\exists \mathbf{x}\, \exists \mathbf{y}\, \mathbf{on}(\mathbf{x}, \mathbf{y}) \land \mathbf{green}(\mathbf{x}) \land \neg \mathbf{green}(\mathbf{y})$$

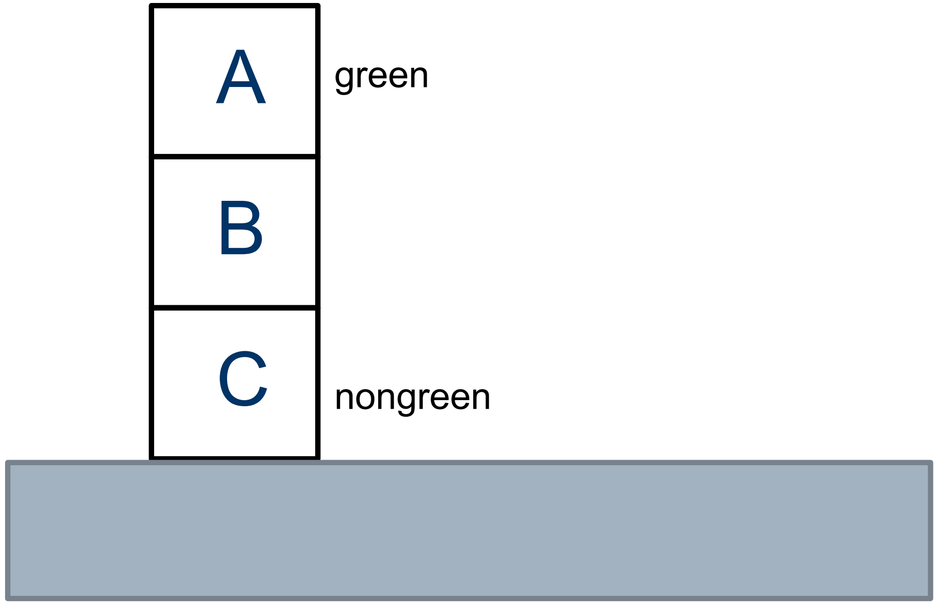
Resolution Refutation Proofs

Example 3

Negation of the Query

$$\neg\exists x\, \exists y\, (on(x, y) \wedge green(x) \wedge \neg green(y))$$

C5. $\neg on(x,y) \vee \neg green(x) \vee green(y)$



Refutation Trace

1.	$on(A,B)$	C1
2.	$on(B,C)$	C2
3.	$green(A)$	C3
4.	$\neg green(C)$	C4
5.	$\neg on(x,y) \vee \neg green(x) \vee green(y)$	C5
6.	$\neg green(A) \vee green(B)$	1,5
7.	$\neg green(B) \vee green(C)$	2,5
8.	$green(B)$	3,6
9.	$\neg green(B)$	4,7
10.	$\square$.	8,9

In this problem the KB entails that there is some block which must be green and on top of a nongreen block.

However, it does not make any commitment to any specific one;

A general method that has been proposed for dealing with such situations is answer extraction.