

# Using Propositional Logic

## Representing simple facts

It is raining  
RAINING

It is sunny  
SUNNY

It is windy  
WINDY

If it is raining, then it is not sunny  
 $\text{RAINING} \rightarrow \neg \text{SUNNY}$

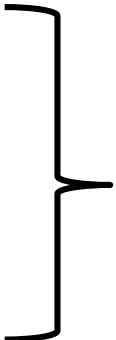
# Normal Forms in propositional Logic

1. Conjunctive normal form (CNF):

$$\text{e.g. } (P \vee Q \vee R) \wedge (P \vee Q) \wedge (P \vee R) \wedge P$$

It is conjunction ( $\wedge$ ) of disjunctions ( $\vee$ )

Where disjunctions are:

1.  $(P \vee Q \vee R)$
  2.  $(P \vee Q)$
  3.  $(P \vee R)$
  4.  $P$
- 
- clauses

## 2. Disjunctive normal form (DNF):

e.g.  $(P \wedge Q \wedge R) \vee (P \wedge Q) \vee (P \wedge R) \vee P$

It is disjunction ( $\vee$ ) of conjunctions ( $\wedge$ )

# Procedure to convert a statement to CNF

**1. Eliminate implications and biconditionals using formulas:**

- $(P \Leftrightarrow Q) \Rightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$
- $P \rightarrow Q \Rightarrow \neg P \vee Q$

**2. Apply De-Morgan's Law and reduce NOT symbols so as to bring negations before the atoms. Use:**

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

**3. Use distributive and other laws & equivalent formulas to obtain Normal forms.**

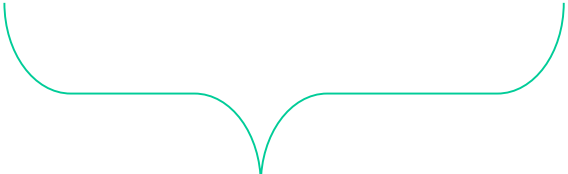
# Conversion to CNF example

Q. Convert into CNF :  $((P \rightarrow Q) \rightarrow R)$

Solution:

$$\begin{aligned} \text{Step 1: } ((P \rightarrow Q) \rightarrow R) & \implies ((\neg P \vee Q) \rightarrow R) \\ & \implies \neg(\neg P \vee Q) \vee R \end{aligned}$$

$$\text{Step 2: } \neg(\neg P \vee Q) \vee R \implies (P \wedge \neg Q) \vee R$$

$$\text{Step 3: } (P \wedge \neg Q) \vee R \implies (P \vee R) \wedge (\neg Q \vee R)$$


CNF

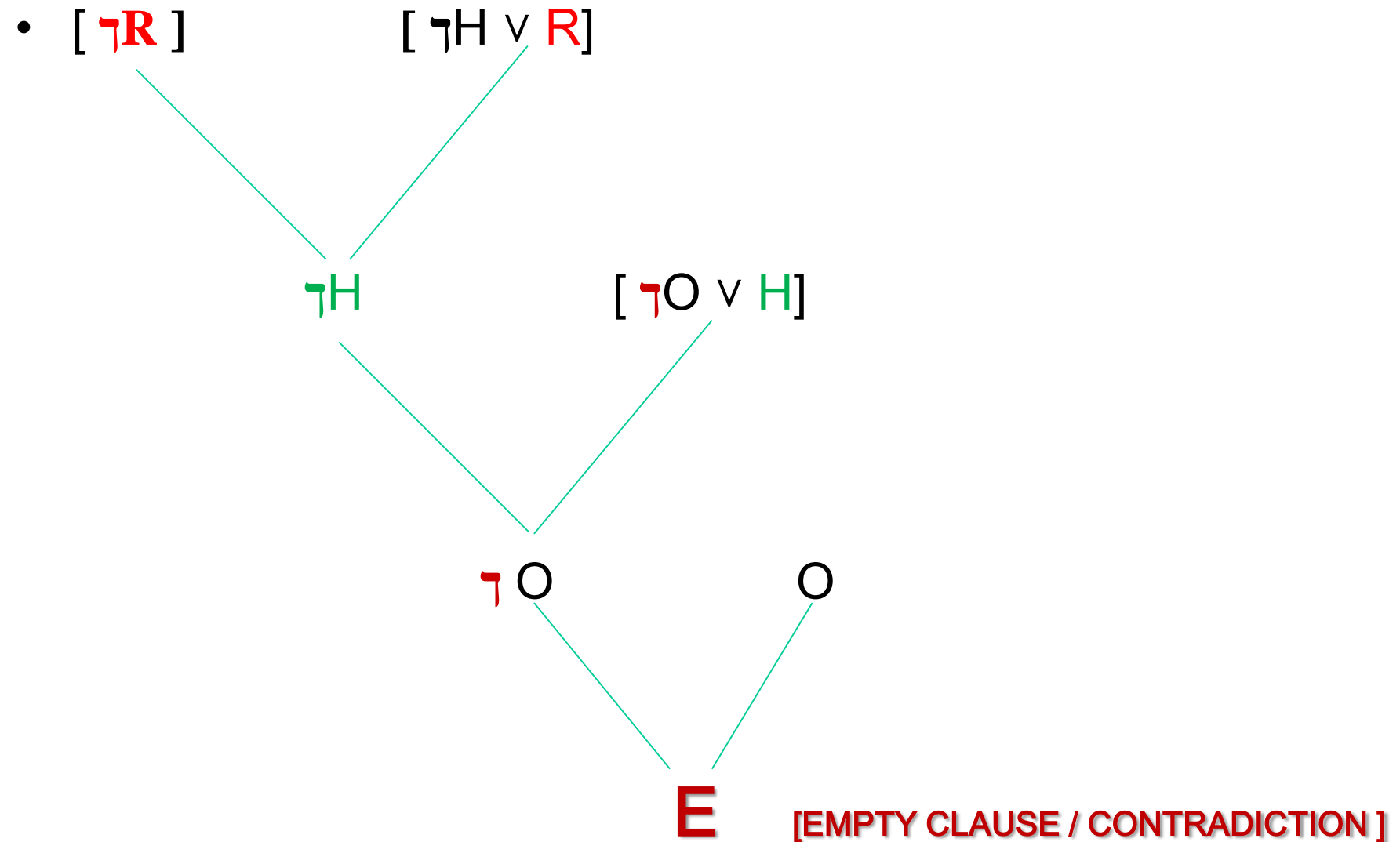
# Resolution in propositional logic

## Proof by Refutation / contradiction.

- Used for theorem proving / rule of inference.
- Method: Say we have to prove proposition A
- Assume A to be false i.e.  $\neg A$
- Continue solving the algorithm starting from  $\neg A$
- If you get a contradiction (F) at the end it means your initial assumption i.e.  $\neg A$  is false and hence proposition A must be true.
- **Clause**: disjunction of literals is called clause.

- How it works?
- E.g. “ If it is Hot then it is Humid. If it is humid then it will rain. It is hot.” prove that “ it will rain.”
- Solution:
- Let us denote these statements with propositions H,O and R:
  - H: “ It is humid”.
  - O: “ It is Hot”. And R: “It will rain”.
- Formulas corresponding to the sentences are:
- 1. “if it is hot then it is humid”       $[O \rightarrow H] \implies \neg O \vee H$
- 2. “If it is humid then it will rain”.  $[H \rightarrow R] \implies \neg H \vee R$
- 3. “ It is Hot”       $[O] \implies O$
- 
- To prove: R.

- Let us assume “it will NOT rain”     $[\neg R]$





- Since an empty clause (  $E$  ) has been deduced we say that our assumption is wrong and hence we have proved:  
“It will rain”

## Using Propositional Logic:

- Theorem proving is decidable BUT
- It Cannot represent objects and quantification.
- Hence we go for PREDICATE LOGIC

# PREDICATE LOGIC

- Can represent **objects** and **quantification**
- Theorem proving is **semi-decidable**

## Representing simple facts (Preposition)

"SOCRATES IS A MAN"

SOCRATESMAN -----1

"PLATO IS A MAN"

PLATOMAN -----2

Fails to capture relationship between Socrates and man.  
We do not get any information about the objects involved  
Ex:

if asked a question : "who is a man?" we cannot get answer.

Using Predicate Logic however we can represent above facts as:    Man(Socretes)    and    Man(Plato)

# Using Predicate Logic

1. Marcus was a man.

`man(Marcus)`

# Using Predicate Logic

2. Marcus was a Pompeian.

Pompeian(Marcus)

- Quantifiers:
- 2 types:-
- Universal quantifier ( $\forall$ )
- $\forall x$ : means “for all” x
- It is used to represent phrase “for all”.
- It says that something is true for all possible values of a variable.
- Ex. “John loves everyone”
-

- Quantifiers:
- 2 types:-
- Universal quantifier ( $\forall$ )
- $\forall x$ : means “for all” x
- It is used to represent phrase “for all”.
- It says that something is true for all possible values of a variable.
- Ex. “John loves everyone”  
 $\forall x: \text{loves}(\text{John}, x)$

- Existential quantifier (  $\exists$  ):
- Used to represent the fact “there exists some”
- Ex:
- “some people like reading and hence they gain good knowledge”

$\exists x: \{ [\text{person}(x) \wedge \text{like}(x, \text{reading})] \rightarrow \text{gain}(x, \text{knowledge}) \}$

- “lord Haggins has a crown on his head”
- $\exists x: \text{crown}(x) \wedge \text{onhead}(x, \text{Haggins})$



# Nested Quantifiers

- We can use both  $\forall$  and  $\exists$  separately
- Ex: “everybody loves somebody”

$$\forall x: \exists y: \text{loves} (x, y)$$

- Connection between  $\forall$  and  $\exists$
- “everyone dislikes garlic”

$$\forall x: \neg \text{like} (x, \text{garlic})$$

➤ This can be also said as:

“there does not exists someone who likes garlic”

$$\neg \exists x: \text{like} (x, \text{garlic})$$

3. All Romans were either loyal to Caesar or hated him.

$$\forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})$$

4. Every one is loyal to someone.

$$\forall x: \exists y: \text{loyalto}(x, y)$$

$$\exists y: \forall x: \text{loyalto}(x, y)$$

5. People **only** try to assassinate rulers they are not loyal to.

$$\begin{aligned} \forall x: \forall y: \text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y) \\ \rightarrow \neg \text{loyalto}(x, y) \end{aligned}$$

6. “All Pompeians were Romans”

$\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x)$

8. Marcus tried to assassinate Caesar.

$\text{tryassassinate}(\text{Marcus}, \text{Caesar})$

# Some more examples

- “all indoor games are easy”

$$\forall x: \text{indoor\_game}(x) \rightarrow \text{easy}(x)$$

- “Rajiv likes only cricket”

$$\text{Like}(\text{Rajiv}, \text{Cricket})$$

- “Any person who is respected by every person is a king”

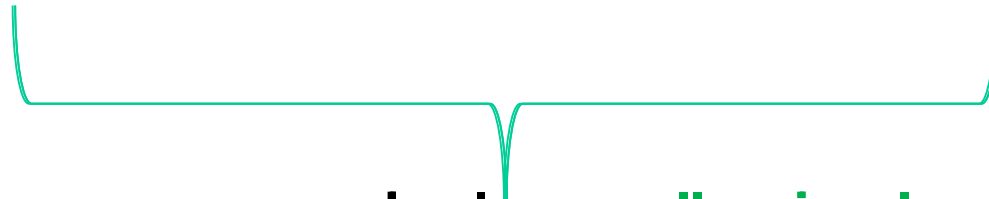
$$\exists x: \forall y: \{ \text{person}(x) \wedge \text{person}(y) \wedge \text{respects}(y, x) \rightarrow \text{king}(x) \}$$

- “god helps those who helps themselves”

$\forall x: \text{helps}(\text{god}, \text{helps}(x, x))$

- “everyone who loves all animals is loved by someone”

$\forall x: [ \forall y: \text{animal}(y) \rightarrow \text{loves}(x, y) ]$



everyone who loves all animals

$\exists z: \text{loves}(z, x)$  } there exist someone z and z loves x

Thus the predicate sentence is:

$\forall x: [ [ \forall y: \text{animal}(y) \rightarrow \text{loves}(x, y) ] \rightarrow [ \exists z: \text{loves}(z, x) ] ]$

# Computable functions and predicates

- “ Marcus was born in 40 A.D”  
Born( Marcus, 40)
- “ All Pompeians died when volcano erupted in 79 A.D”  
Erupted(volcano, 79)  $\wedge \forall x: [ \text{Pompeian} (x) \rightarrow \text{Died} (x, 79)]$
- “ no mortal lives longer than 150 years”
- How to solve ?
- let **t1 is time instance 1** and **t2 is time instance 2**
- We use computable function **gt( ... , ....)** which computes greater than.

$$\forall x: \forall t1: \forall t2: \text{mortal} (x) \wedge \text{born} (x, t1) \wedge \text{gt}( t2 - t1, 150 ) \rightarrow \text{dead} (x, t2)$$

# Resolution algorithm in predicate logic

- Proof by refutation.
- INPUT: Predicate sentences in clausal form (CNF)
- (See conversion algo on next slide)
- Algorithm steps :-

Convert all the propositions of KB to clause form (S).

2. Negate  $\alpha$  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  $(\alpha \vee \neg P)$  and  $(\gamma \vee P)$ .
  - b. Add the resolvent  $(\alpha \vee \gamma)$  to S.

# Conversion to Clause Form

1. Eliminate  $\rightarrow$ .

$$P \rightarrow Q \equiv \neg P \vee Q$$

2. Reduce the scope of each  $\neg$  to a single term.

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

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

$$\neg \forall x: P \equiv \exists x: \neg P$$

$$\neg \exists x: p \equiv \forall x: \neg p$$

$$\neg \neg P \equiv P$$

3. Standardize **variables** so that each quantifier binds a unique variable.

$$(\forall x: P(x)) \vee (\exists x: Q(x)) \equiv$$

$$(\forall x: P(x)) \vee (\exists y: Q(y))$$



4. Move all **quantifiers** to the left without changing their relative order.

$$(\forall \mathbf{x}: P(\mathbf{x})) \vee (\exists \mathbf{y}: Q(\mathbf{y})) \equiv \\ \forall \mathbf{x}: \exists \mathbf{y}: (P(\mathbf{x}) \vee (Q(\mathbf{y})))$$

5. Eliminate  $\exists$  (Skolemization).

$$\exists \mathbf{x}: P(\mathbf{x}) \equiv P(\mathbf{c}) \quad \text{Skolem constant}$$

$$\forall \mathbf{x}: \exists \mathbf{y} P(\mathbf{x}, \mathbf{y}) \equiv \forall \mathbf{x}: P(\mathbf{x}, \mathbf{f}(\mathbf{x})) \quad \text{Skolem function}$$

6. Drop  $\forall$ .

$$\forall \mathbf{x}: P(\mathbf{x}) \equiv P(\mathbf{x})$$

7. Convert the formula into a **conjunction of disjuncts**.

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

8. Create a **separate clause** corresponding to each conjunct.

9. Standardize apart the **variables** in the set of obtained clauses.

- Example of conversion:

$$\forall x: \neg [ \text{Roman}(x) \rightarrow ( \text{Pompeian}(x) \wedge \neg \text{hate}(x, \text{Caesar}) ) ]$$

After step 1: i.e. elimination of  $\rightarrow$  and  $\Leftrightarrow$  the above stmt becomes:

$$\forall x: \neg [ \neg \text{Roman}(x) \vee ( \text{Pompeian}(x) \wedge \neg \text{hate}(x, \text{Caesar}) ) ]$$

After step 2: i.e. reducing scope of  $\neg$  the above stmt becomes:

$$\forall x: [ \text{Roman}(x) \wedge \neg ( \text{Pompeian}(x) \wedge \neg \text{hate}(x, \text{Caesar}) ) ]$$



$$\forall x: [ \text{Roman}(x) \wedge ( \neg \text{Pompeian}(x) \vee \text{hate}(x, \text{Caesar}) ) ]$$

- Example to demonstrate step 3:- i.e. standardization of variables.

$$\forall x: [ [\forall y: \text{animal}(y) \rightarrow \text{loves}(x, y)] \rightarrow [\exists y: \text{loves}(y, x)] ]$$

After step 3 above stmt becomes,

$$\forall x: [ [\forall y: \text{animal}(y) \rightarrow \text{loves}(x, y)] \rightarrow [\exists z: \text{loves}(z, x)] ]$$

- Example to demonstrate step 4: Move all quantifiers to the left without changing their relative order.
- 
- $\forall x: [ [ \forall y: \text{animal}(y) \wedge \text{loves}(x, y) ] \vee [ \exists z: \text{loves}(z, x) ] ]$
- After applying step 4 above stmt becomes:
- $\forall x: \forall y: \exists z: [ \text{animal}(y) \wedge \text{loves}(x, y) \vee \text{loves}(z, x) ]$
- After first 4 processing steps of conversion are carried out on original statement S, the statement is said to be in **PRENEX NORMAL FORM**

- Example to demonstrate step 5: skolemization ( i.e. elimination of  $\exists$  quantifier )

$\exists y: \text{President}(y)$

Can be transformed into

President (**S1**)

where **S1** is a function that somehow produces a value that satisfies President (S1) – S1 called as **Skolem constant**

- Ex. 2:

$\exists y: \forall x: \text{leads}(y, x)$

Here value of **y** that satisfies 'leads' depends on particular value of **x** hence above stmt can be written as:

$\forall x: \text{leads}(f(x), x)$

Where  $f(x)$  is **skolem function**.

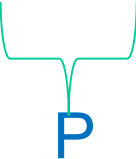
- Example to demonstrate step 6: dropping prefix  $\forall$

$$\forall x: \forall y: \forall z: [\neg \text{Roman}(x) \vee \neg \text{know}(x, y) \vee \text{hate}(y, z)]$$

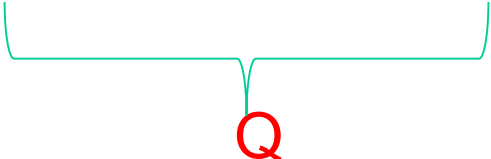
- After prefix dropped becomes,

$$[\neg \text{Roman}(x) \vee \neg \text{know}(x, y) \vee \text{hate}(y, z)]$$


- Example to demonstrate step 7: Convert the formula into a **conjunction of disjuncts**.(CNF)
- $\text{Roman}(x) \vee ( (\text{hate}(x, \text{caesar}) \wedge \neg \text{loyalto}(x, \text{caesar}) )$
- $\text{Roman}(x) \vee ( (\text{hate}(x, \text{caesar}) \wedge \neg \text{loyalto}(x, \text{caesar}) )$ 



P



Q



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

**CLAUSE 1** —  $( \text{Roman}(x) \vee ( \text{hate}(x, \text{caesar}) ) \wedge$

**CLAUSE 2** —  $( \text{Roman}(x) \vee \neg \text{loyalto}(x, \text{caesar}) )$



# Unification

- It's a matching procedure that compares two literals and discovers whether there exists a set of substitutions that can make them identical.

- E.g.

Hate( marcus , X)

Hate (marcus , caesar)

caesar/ X



e.g. 2.

Hate(X,Y)   Hate( john, Z)   could be unified as:

John/X   and   y/z

## Unification:

$\text{UNIFY}(p, q) = \text{unifier } \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$

$\forall x: \text{knows}(\text{John}, x) \rightarrow \text{hates}(\text{John}, x)$

$\text{knows}(\text{John}, \text{Jane})$

$\forall y: \text{knows}(y, \text{Leonid})$

$\forall y: \text{knows}(y, \text{mother}(y))$

$\forall x: \text{knows}(x, \text{Elizabeth})$

$\text{UNIFY}(\text{knows}(\text{John}, \underline{x}), \text{knows}(\text{John}, \underline{\text{Jane}})) = \{\text{Jane}/x\}$

$\text{UNIFY}(\text{knows}(\underline{\text{John}}, \underline{x}), \text{knows}(y, \underline{\text{Leonid}})) = \{\text{Leonid}/x, \text{John}/y\}$

$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(y, \text{mother}(y))) = \{\text{John}/y, \text{mother}(\text{John})/x\}$

$\text{UNIFY}(\text{knows}(\text{John}, x), \text{knows}(x, \text{Elizabeth})) = \text{FAIL}$

# Resolution algorithm

- It is used as inference mechanism.
- Pre-processing steps:
  1. Convert the given English sentence into predicate sentence.
  2. Not all of these sentences will be in clausal form (CNF).  
If any sentence is not in clausal form then convert it into clausal form.
  3. Give these sentences (clauses) as an **input** to resolution algorithm.

## Resolution algorithm steps:

A. Negate the proposition which is to be proved.

i.e. If we have to prove :-

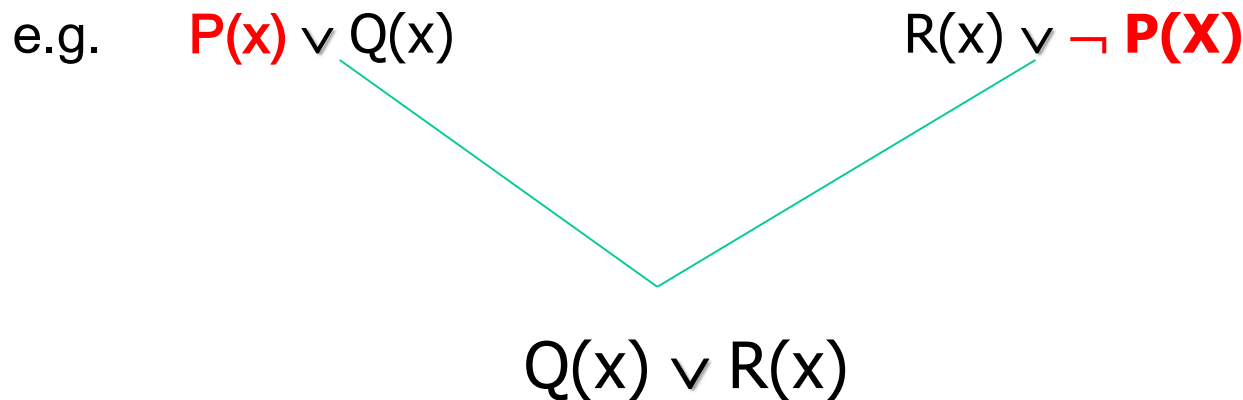
like(tommy , cookies) then assume  $\neg$  like(tommy,cookies)

Add the resultant sentence to the set of sentences from step 3

B. Repeat until contradiction is found or no progress can be made:

- i. Select two clauses, call them parent clauses and resolve them together.

The resultant clause is called resolvent.



- ii. If resolvent contains empty clause then **contradiction** has been found.



- iii. If step ii. Results in empty clause , it means our assumption is wrong and the original clause (to be proved) has to be true.

# Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

1. “Marcus was a man”

man(marcus)

----- 1

2. “Marcus was a Pompeian”

pompeian (marcus)

----- 2

3. “All Pompeian's were Romans”

$\Rightarrow \forall x1: \text{pompeian}(x1) \rightarrow \text{roman}(x1).$

$\Rightarrow \forall x1: \neg \text{pompeian}(x1) \vee \text{roman}(x1)$

$\neg \text{pompeian}(x1) \vee \text{roman}(x1)$

----- 3

4. “Caesar was a ruler”

ruler (caesar)

----- 4

5. “all romans were either loyalto caesar or hated him”

$\Rightarrow \forall x2: \text{roman}(x2) \rightarrow [\text{loyalto}(x2, \text{caesar}) \vee \text{hate}(x2, \text{caesar})]$

$\Rightarrow \forall x2: \neg \text{roman}(x2) \vee \text{loyalto}(x2, \text{caesar}) \vee \text{hate}(x2, \text{caesar})$

$\Rightarrow \neg \text{roman}(x2) \vee \text{loyalto}(x2, \text{caesar}) \vee \text{hate}(x2, \text{caesar})$

$\neg \text{roman} ( x2) \vee \text{loyalto} (x2, \text{caesar}) \vee \text{hate} (x2, \text{caesar})$

----- 5

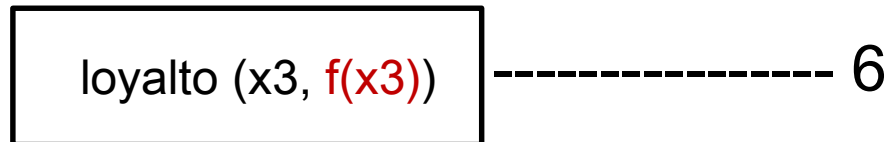


- “Every one is loyal to someone”  
 $\Rightarrow \forall x3: \exists y1: \text{loyalto}(x3, y1).$

Let  $f(x3)$  be a skolem function then,

$\Rightarrow \forall x3: \text{loyalto}(x3, f(x3)).$

$\Rightarrow \text{loyalto}(x3, f(x3))$



7. “People only try to assassinate rulers they are not loyal to.”

$$\Rightarrow \forall x_4: \forall y_2: [\text{man}(x_4) \wedge \text{ruler}(y_2) \wedge \text{tryassassinate}(x_4, y_2)] \\ \rightarrow \neg \text{loyalto}(x_4, y_2)$$

$$\Rightarrow \forall x_4: \forall y_2: \neg [\text{man}(x_4) \wedge \text{ruler}(y_2) \wedge \text{tryassassinate}(x_4, y_2)] \\ \vee \neg \text{loyalto}(x_4, y_2)$$

$$\Rightarrow \forall x_4: \forall y_2: \neg \text{man}(x_4) \vee \neg \text{ruler}(y_2) \vee \neg \text{tryassassinate}(x_4, y_2) \vee \\ \neg \text{loyalto}(x_4, y_2)$$

let  $f(x_4)$  be skolem function then,

$$\Rightarrow \Rightarrow \forall x_4: \neg \text{man}(x_4) \vee \neg \text{ruler}(f(x_4)) \vee \\ \neg \text{tryassassinate}(x_4, f(x_4)) \vee \neg \text{loyalto}(x_4, f(x_4))$$

$\Rightarrow \neg \text{man}(x4) \vee \neg \text{ruler}(f(x4)) \vee \neg \text{tryassassinate}(x4, f(x4)) \vee \neg \text{loyalto}(x4, f(x4))$

$\neg \text{man}(x4) \vee \neg \text{ruler}(f(x4)) \vee \neg \text{tryassassinate}(x4, f(x4)) \vee \neg \text{loyalto}(x4, f(x4))$

----- 7

8. “Marcus tried to assassinate Caesar”

`tryassassinate(marcus , caesar)`

`tryassassinate( marcus , caesar )`

----- 8

To prove : marcus hate caesar      i.e.    `hate(marcus, caesar)`

- Assume  $\neg \text{hate}(\text{marcus}, \text{caesar})$

(5)

$\neg \text{hate}(\text{marcus}, \text{caesar})$

$\neg \text{roman}(x_2) \vee \text{loyalto}(x_2, \text{caesar}) \vee$   
 $\text{hate}(x_2, \text{caesar})$

$x_2 / \text{marcus}$

$\neg \text{roman}(\text{marcus}) \vee \text{loyalto}(\text{marcus}, \text{caesar})$

$\neg \text{pompeian}(x_1) \vee \text{roman}(x_1)$

(3)

$x_1 / \text{marcus}$

(2)

$\text{pompeian}(\text{marcus})$

$\neg \text{pompeian}(\text{marcus}) \vee \text{loyalto}(\text{marcus}, \text{caesar})$

$\text{loyalto}(\text{marcus}, \text{caesar})$

loyalto (marcus, caesar)

(7)

$\neg \text{man}(x4) \vee \neg \text{ruler}(f(x4)) \vee \neg \text{tryassassinate}(x4, f(x4)) \vee$   
 $\neg \text{loyalto}(x4, f(x4))$

x4/ marcus  
f(x4)/ caesar

$\neg \text{man}(\text{marcus}) \vee \neg \text{ruler}(\text{caesar}) \vee \neg \text{tryassassinate}(\text{marcus}, \text{caesar})$

(8)

tryassassinate( marcus , caesar )

$\neg \text{man}(\text{marcus}) \vee \neg \text{ruler}(\text{caesar})$

(1)

man( marcus)

$\neg \text{ruler}(\text{caesar})$

$\neg \text{ruler}(\text{caesar})$

(4)

$\text{ruler}(\text{caesar})$

**E**

- Since we get an empty clause i.e. contradiction our assumption that  $\neg \text{hate}(\text{marcus}, \text{caesar})$  is false

hence

$\text{hate}(\text{marcus}, \text{caesar})$  must be true.

- Consider the following paragraph:

“ anything anyone eats is called food. Milka likes all kind of food. Bread is a food. Mango is a food. Alka eats pizza. Alka eats everything milka eats.”

Translate the following sentences into (WFF) in predicate logic and then into set of clauses. Using resolution principle answer the following:

1. Does Milka like pizza?

2. what food Alka eats?      [ Question answering]

- Solution:

1. “ anything anyone eats is called food.”

$$\forall x: \forall y: \text{eats}(x, y) \rightarrow \text{food}(y)$$

$$\Rightarrow \forall x: \forall y: \neg \text{eats}(x, y) \vee \text{food}(y)$$

$$\Rightarrow \neg \text{eats}(x, y) \vee \text{food}(y) \quad (1)$$

2. “Milka likes all kind of food”

$$\forall y1: \text{food}(y1) \rightarrow \text{like}(\text{milka}, y1)$$

$$\Rightarrow \forall y1: \neg \text{food}(y1) \vee \text{like}(\text{milka}, y1)$$

$$\Rightarrow \neg \text{food}(y1) \vee \text{like}(\text{milka}, y1) \quad (2)$$

3. “Bread is a food”

$$\text{food}(\text{bread}) \quad (3)$$

4. “Mango is a food”

$$\text{food}(\text{mango}) \quad (4)$$



5. “Alka eats Pizza”

$\text{eats}(\text{alka}, \text{pizza})$  (5)

6. “Alka eats everything Milka eats”

$\forall x1: \text{eats}(\text{milka}, x1) \rightarrow \text{eats}(\text{alka}, x1)$

$\Rightarrow \forall x1: \neg \text{eats}(\text{milka}, x1) \vee \text{eats}(\text{alka}, x1)$

$\Rightarrow \neg \text{eats}(\text{milka}, x1) \vee \text{eats}(\text{alka}, x1)$  (6)

Question to be answered : 1. “Does Milka likes Pizza ?”

assume : “Milka does not like Pizza”

$\neg \text{like}(\text{milka}, \text{pizza})$  (7)

$\neg \text{like}(\text{milka}, \text{pizza})$

(2)

$\neg \text{food}(\text{y1}) \vee \text{like}(\text{milka}, \text{y1})$

$\text{pizza}/\text{y1}$

$\neg \text{food}(\text{pizza})$

$\text{pizza}/\text{y}$

(1)

$\text{eats}(\text{x}, \text{y}) \vee \text{food}(\text{y})$

$\neg \text{eats}(\text{x}, \text{pizza})$

(5)

$\text{eats}(\text{alka}, \text{pizza})$

$\text{alka}/\text{x}$

**E**

Since  $\neg \text{like}(\text{milka}, \text{pizza})$  is contradiction  $\text{like}(\text{milka}, \text{pizza})$  is **true**

Question to be answered : 2. “ what food Alka eats ?”

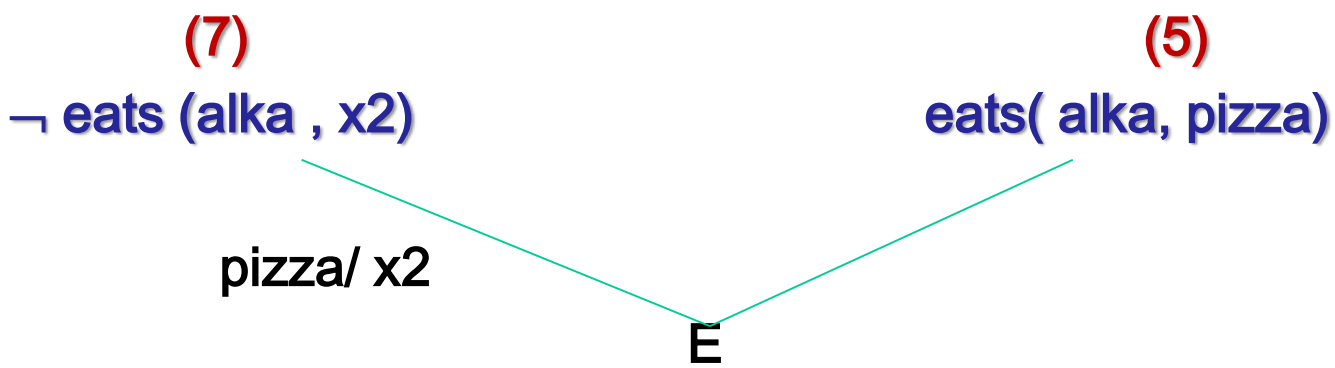
eats( alka, ??)

there exist something which Alka eats we have to find the value of x

$\exists x: \text{ eats ( alka, x)}$

Assume : alka does not eat anything

$\neg [\exists x_2: \text{ eats ( alka, x}_2)]$   
=>  $\forall x_2: \neg \text{ eats ( alka , x}_2)$   
=>  $\neg \text{ eats ( alka , x}_2)$  (7)



- Therefore alka does not eat anything is false and
- Alka eats something is true.
- And x2 stores pizza
- Therefore we conclude :

eats ( alka, ??) answer is “pizza”

# Instance and Isa relationship

- “ Marcus is a man”

man(marcus)

OR

**instance**( marcus , man)

where marcus is an object/  
instance of class ‘man’

“ all pompeians were romans”

$\forall x: \text{pompeian}(x) \rightarrow \text{roman}(x).$

OR

$\forall x: \text{instance}(x, \text{pompeian}) \rightarrow \text{instance}(x, \text{roman}).$

- Isa Predicate :

“ all pompeians were romans”

$\forall x: \text{pompeian}(x) \rightarrow \text{roman}(x).$

OR

$\forall x: \text{instance}(x, \text{pompeian}) \rightarrow \text{instance}(x, \text{roman}).\text{-----}(1)$

- Now using isa predicate (1) becomes,

$\text{isa}(\text{pompeian}, \text{roman})$

which means pompeian is a subclass of roman class  
but it also requires extra axiom :

$\forall x: \forall y: \forall z: \text{isa}(y, z) \wedge \text{instance}(x, y) \rightarrow \text{instance}(x, z)$

# Using Predicate Logic

- Many English sentences are **ambiguous**.
- There is often a **choice** of how to represent knowledge.
- **Obvious information** may be necessary for reasoning
- We may not know in advance which **statements to deduce** ( $P$  or  $\neg P$ ).

THANK YOU

