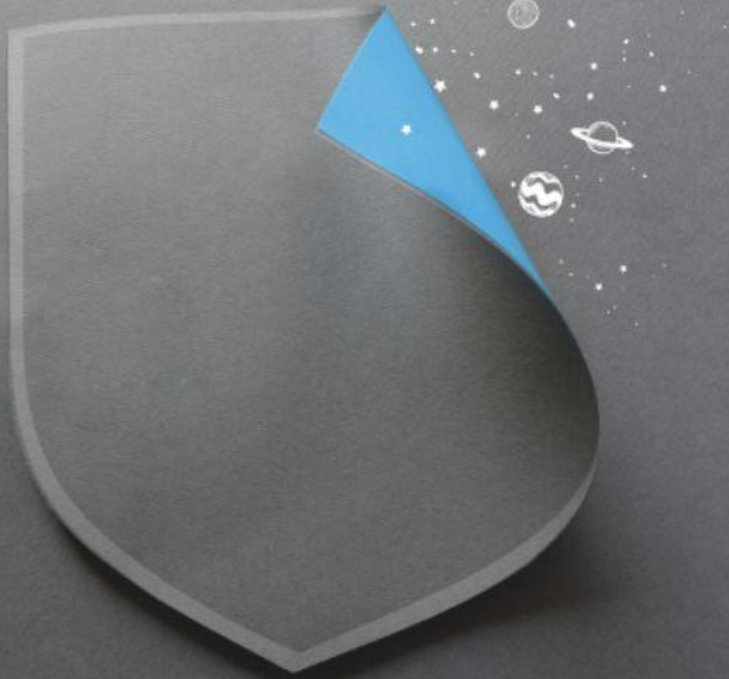
Artificial Intelligence

Unit-4 (Predicate Logic, Resolution)

Artificial Intelligence 01CE0702



Marwadi University

Department of Computer Engineering

Shilpa Singhal

Limitations of Propositional Logic

- The propositional logic has its limitations that you cannot deal properly with general statements of the form
- “All men are mortal”
- You can not derive from the conjunction of this and “Socrates is a man” that “Socrates is mortal”

Example:

- If All men are mortal = P
- Socrates is a Man = Q
- Socrates is mortal = R
- Then (P & Q) $\rightarrow$ R is not valid

First Order Predicate Logic (FOPL)

- FOPL was developed to extend the expressiveness of Propositional logic.
- Propositional logic works fine in situations where the result is either true or false. However, there are many real life situations that cannot be treated this way.
- Predicate logic is the area of logic that deals with predicates and quantifiers
- Predicate – refers to a property that the subject of a statement can have

E.G. "x is greater than 3"

x → is the subject and

"is greater than 3" → is the predicate

Syntax of FOPL

- Connectives
- Quantifiers
- Constants
- Variables
- Predicates
- Functions

Syntax of FOPL

- **Connectives**: There are five connective symbols
  - ~ : not or negation
  - & : and or conjuction
  - $\vee$: or or inclusive disjunction
  - $\rightarrow$: implication
  - $\leftrightarrow$: equivalence or if and only if

- **Quantifiers**: There are two quantifier symbols are
  - $\exists$: existential quantification
    - ➢Example: $\exists$x means **there exist** x
  - $\forall$: universal quantification
    - ➢Example: $\forall$x means **for all** x

Syntax of FOPL...

- **Constants**: constants are fixed-values terms that belong to a given domain of discourse. They are denoted by numbers, words and small letter.
- **Variables:** variables are terms that can assume different values over a given domain.

Syntax of FOPL...

- **Predicates:** a predicate is defined as a relation that binds two atoms together. Ex: Rabbit likes carrots, is represented as
    - ➢**LIKES(**rabit, carrots)

      Here **LIKES is a predicate that links two atoms “rabbit“ and “carrots“**

- **Functions**: it is also possible to have a function as an argument, e.g. “ Ravi,,s father is Rani,,s father“ is represented as-
    - FATHER(father(Ravi), Rani)
    - Here FATHER is a predicate and father(Ravi) is a function to indicate Ravi,,s father.

Syntax of FOPL...

- Constants, variables, and functions are referred to as **terms**.
- Predicates are referred to as **atomic formulas** or **atoms**.
- When we want to refer to an atom or its negation, we often use the word **literal**.
- In addition to above symbols, left and right parentheses , square brackets, braces, and the period are used for punctuation in symbolic representation.

Translating English to FOPL

1. Bhaskar likes aeroplanes.
2. Ravi‟s father is rani‟s father.
3. Plato is a man
4. Ram likes mango.
5. Sima is a girl.
6. Rose is red.
7. John owns gold
8. Ram is taller than mohan
9. My name is khan
10. Apple is fruit.

Translating English to FOPL

11. Ram is male.
12. Tuna is fish.
13. Dashrath is ram‟s father.
14. Kush is son of ram.
15. Kaushaliya is wife of Dashrath.
16. Clinton is tall.
17. There is a white alligator.
18. All kings are person.
19. Nobody loves john.
20. Every one has a father.

Translating English to FOPL

1. Bhaskar likes aeroplanes.	Likes (bhaskar, aeroplanes).
2. Ravi‟s father is rani‟s father.	Father(father(ravi), rani)).
3. Plato is a man.	Man (plato).
4. Ram likes mango.	Likes(ram, mango).
5. Sima is a girl.	Girl(sima) .
6. Rose is red.	Red (rose).
7. John owns gold.	Owns(john, gold).
8. Ram is taller than mohan.	Taller(ram, mohan).
9. My name is khan.	Name (khan) or Name(my, khan)
10. Apple is fruit.	Fruit(apple).

Translating English to FOPL

11. Ram is male.	Male (ram).
12. Tuna is fish.	Fish (tuna).
13. Dashrath is ram‟s father.	Father (dashrath, ram).
14. Kush is son of ram.	Son(kush, ram).
15. Kaushaliya is wife of Dashrath.	Wife(kaushaliya, dashrath).
16. Clinton is tall.	Tall(clinton).
17. There is a white alligator.	Alligator (white).
18. All kings are person.	$\forall x$: Kings(x) $\rightarrow$ Person(x).
19. Nobody loves john.	$\forall x$: $\neg$Loves(x, john).
20. Every one has a father.	$\forall x$: $\exists$**y: Father(y,x)**

Translate into predicate logic

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they aren't loyal to.
8. Marcus tried to assassinate Caesar.
9. All men are people.

Solution

1. Marcus was a man.
   - Man(Marcus).
2. Marcus was a Pompeian.
   - Pompeian(marcus)
3. All Pompeian were Romans.
   - $\forall$x: Pompeian(x) $\rightarrow$ Roman(x)
4. Caesar was a ruler.
   - Ruler(Caesar)
5. All Romans were either loyal to Caesar or hated him.
   - $\forall$x: Roman(x) $\rightarrow$ LoyalTo(x,Caesar) $\vee$ Hate(x,Caesar)

Solution

6. Everyone is loyal to someone.

$\forall x: \exists y: LoyalTo(x,y)$

7. People only try to assassinate rulers they aren't loyal to.

$\forall x: \forall y: Person(x) \wedge Ruler(y) \wedge TryAssassinate(x,y) \rightarrow \neg LoyalTo(x,y)]$

8.Marcus tried to assassinate Caesar.

TryAssassinate(Marcus, Caesar)

9. All men are people.

$\forall x: Men(x) \rightarrow People(x)$

Translate into predicate logic

i. Hari likes all kind of food.
ii. Bananas are food.
iii. Apples are food.
iv. Anything anyone eats and isn‟t killed by food.
v. Hari eats everything ram eats.

Solution

ι. $\forall x$: Food(x) $\rightarrow$ Likes(hari, x).

ii. Food(bananas) .

iii. Food(apples) .

ιϖ. $\forall x$: $\exists y$: Eats(y,x) ^ $\neg$Killedby(y,x) $\rightarrow$Food(x)

ϖ. $\forall x$: eats(ram, x) $\rightarrow$ eats(hari, x) .

Translating English to FOPL

1. Every gardener likes the sun.
2. Not Every gardener likes the sun.
3. You can fool some of the people all of the time.
4. You can fool all of the people some of the time.
5. You can fool all of the people at same time.
6. You can not fool all of the people all of the time.
7. Everyone is younger than his father.

Solution

1. (∀x): gardener(x) => likes(x, Sun)
2. ~((∀x) :gardener(x) => likes(x, Sun))
3. (∃x):(∀t) :person(x) ^ time(t) => can-be-fooled(x, t)
4. (∀x):(∃t) :person(x) ^ time(t) => can-be-fooled(x ,t)
5. (∃t):(∀x) :person(x) ^ time(t) => can-be-fooled(x, t)
6. ~((∀x):(∀t): person(x) ^ time(t) => can-be-fooled(x, t))
7. (∀x) :person(x) => younger(x, father(x))

Translating English to FOPL

1. All purple mushrooms are poisonous.
2. No purple mushroom is poisonous.
3. There are exactly two purple mushrooms.
4. Clinton is not tall.
5. X is above Y if X is directly on top of Y or there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.
6. no one likes everyone

Solution

1. (∀x): (mushroom(x) ^ purple(x)) => poisonous(x)
2. ~(∃x): purple(x) ^ mushroom(x) ^ poisonous(x)

   (∀x): (mushroom(x) ^ purple(x)) => ~poisonous(x)
3. (∃x):(∃y): mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^
4. ~tall(Clinton)
5. (∀z) :(mushroom(z) ^ purple(z)) => ((x=z) v (y=z))

   (∀x):(∀y): above(x,y) <=> (on(x,y) v (∃z) (on(x,z) ^ above(z,y)))

   6. ~ (∃x):(∀y):likes(x,y) or (∀x):(∃y):~likes(x,y)

Skolemization

- Skolemization is the process of replacement of existential quantified variable with Skolem function and deletion of the respective quantifiers.
- Skolem function is arbitrary functions which can always assume a correct value required of an existentially quantified variable.
- Example
  - **$\exists x$: President(x)**
  - Can be transformed into the formula
  - **President(P1)**
  - Where P1 is a function with no arguments that somehow produces a value that satisfies president.

Example 1

Everybody loves somebody.

$\forall$x: $\exists$ y: (Person(x) ^ Person(y)) $\rightarrow$Loves(x,y)]

Converted to

$\forall$x: (Person(x) ^ Person f(x)) $\rightarrow$Loves(x,f(x))]

Where f(x) specifies the person that x loves.

Clausal Form

- A formula is said to be in clausal form if it is of the form:

  $\forall x_1 \forall x_2 \ldots \forall x_n [C_1 A C_2 A \ldots A C_k]$. „

- All first-order logic formulas can be converted to clausal form.

Equivalent Logical Expressions

i. ~(~F) = F (Double Negation)

ii. F & G = G & F, F V G = G V F (Commutativity)

iii. (F & G) & H = F & (G & H), (F V G) V H = F V (G V H) (Associativity)

iv. F V (G & H) = (F V G) & (F V H), F & (G V H) = (F & G) V (F & H) (Distributivity)

v. ~(F & G) = ~F V ~G, ~(F V G) = ~F & ~G (De Morgan)

vi. F $\rightarrow$ G = ~F V G

vii. F $\leftrightarrow$ G = (~F V G) & (~G V F)

ϖιιι. $\forall$x F[x] V G = $\forall$x (F[x] V G)

ιξ. $\exists$x F[x] V G = $\exists$x (F[x] V G)

x. $\forall$x F[x] & G = $\forall$x (F[x] & G)

xi. $\exists$x F[x] & G = $\exists$x (F[x] & G)

Equivalent Logical Expressions...

xii. $\sim(\forall x)\ F[x] = \exists x\ (\sim F[x])$

xiii. $\sim(\exists x)\ F[x] = \forall x\ (\sim F[x])$

ξιϖ.$\forall x\ F[x]\ \&\ \forall x\ G[x] = \forall x\ (F[x]\ \&\ G[x])$

xv. $\exists\ x\ F[x]\ \&\ \exists\ x\ G[x] = \exists\ x\ (F[x]\ \&\ G[x])$

Conversion to Clausal form or Conjunctive Normal Form (CNF)

1. Eliminate logical implications, ⇒, using the fact that A ⇒ B is equivalent to ¬A ∨ B.
2. Reduce the scope of each negation to a single term, using the following facts:

   ¬(¬P) = P

   ¬(A ∨ B) = ¬A ∧ ¬B

   ¬(A ∧ B) = ¬A ∨ ¬B

   ¬∀x: P(x) = ∃x: ¬P(x)

   ¬∃x: P(x) = ∀x: ¬P(x)
3. Standardize variables so that each quantifier binds a unique variable.
4. Move all quantifiers to the left, maintaining their order.
5. Eliminate existential quantifiers, using Skolem functions (functions of the preceding universally quantified variables).
6. Drop the prefix; assume universal quantification.
7. Convert the matrix into a conjunction of disjunctions. [(a &b) or c=(a or c) & (b or c)
8. Create a separate clause corresponding to each conjunction.
9. Standardize apart the variables in the clauses.

Example of Clausal Conversion

- **$\exists x\ \forall y\ (\forall z\ P(f(x), y, z) \rightarrow (\exists u\ Q(x, u)\ \&\ \exists v\ R(y, v)))$**
- **Step 1:** Eliminate logical implications
  - $\exists x\ \forall y\ (\sim(\forall z)\ P(f(x), y, z)\ V\ (\exists u\ Q(x, u)\ \&\ (\ \exists v)\ R(y, v)))$
- **Step 2:** Reduce the scope of each negation to a single term
  - $\exists x\ \forall y\ (\exists z \sim P(f(x), y, z)\ V\ (\exists u\ Q(x, u)\ \&\ (\exists v)\ R(y, v)))$
- **Step 3:**
  - It is not require here because quantifiers have different variable assignments.
- **Step 4:** Move all quantifiers to the left
  - $\exists x\ \forall y\ \exists z\ \exists u\ \exists v\ (\sim P(f(x), y, z)\ V\ (Q(x, u)\ \&\ R(y, v)))$
- **Step 5:** Skolem functions
  - $\forall y(\sim P\ f(x)\ , y, g(y))\ V\ Q(a, h(y))\ \&\ R(y, l(y))$
- **Step 6:** Drop the prefix
  - $(\sim P\ f(x)\ , y, g(y))\ V\ Q(a, h(y))\ \&\ R(y, l(y))$

Example of Clausal Conversion

- **Step 7:** Convert the matrix into a conjunction of disjunctions
  - ((~P f(x) , y, g(y)) V Q(a, h(y)) &(~P f(x) , y, g(y)) V R(y, l(y)))
- **Step 8:**
  - (~P f(x) , y, g(y)) V Q(a, h(y))
  - (~P f(x) , y, g(y)) V R(y, l(y)))
- **Step 9:** Standardize apart the variables in the clauses

Translate into Clausal form

i. Hari likes all kind of food.
ii. Bananas are food.
iii. Apples are food.
iv. Anything anyone eats and isn‟t killed by food.
v. Hari eats everything ram eats.

Convert it into clausal form

ι. $\forall x$: Food(x) $\rightarrow$ Likes(hari, x).

ii. Food(bananas) .

iii. Food(apples) .

ιϖ. $\forall x$: $\exists y$: Eats(y,x) ^ $\neg$Killedby(y,x) $\rightarrow$Food(x)

ϖ. $\forall x$: Eats(ram, x) $\rightarrow$ Eats(hari, x) .

Solution

i. ¬ Food(x) ∨ Likes(hari, x).
ii. Food(bananas) .
iii. Food(apples) .
iv. ¬ Eats(y,x) ∨ Killedby(y,x) ∨ Food(x)
v. ¬ Eats(ram, x) ∨ eats(hari, x) .

Convert it into FOPL

i. All lectures are determined.

$\forall x$: Lecturer(x) $\rightarrow$ Determined(x)

ii. Any one who is determined and intelligent will give good service.

$\forall x$: Determined(x) ^ Intelligent(x)$\rightarrow$Givegoodservice(x)

iii. Mary is an intelligent lecturer.

Lecturer(mary)

Intelligent(mary)

Convert FOPL into clausal form

i. All lectures are determined.

$\forall x$: Lecturer(x) $\rightarrow$ Determined(x)

$\neg$ Lecturer(x) $\vee$ Determined(x)

i. Any one who is determined and intelligent will give good service.

$\forall x$: Determined(x) $\wedge$ Intelligent(x)$\rightarrow$Givegoodservice(x)

$\neg$ Determined(x) $\vee$ $\neg$ Intelligent(x) $\vee$ Givegoodservice(x)

iii. Mary is an intelligent lecturer.

Lecturer(mary)

Intelligent(mary)

(Both are same)

Example of FOPL

1. Marcus was a man.
   - ➢ Man(Marcus).
2. Marcus was a Pompeian.
   - ➢ Pompeian(marcus)
3. All Pompeian were Romans.
   - ➢ $\forall$x: Pompeian(x) $\rightarrow$ Roman(x)
4. Caesar was a ruler.
   - ➢ Ruler(Caesar)
5. All Romans were either loyal to Caesar or hated him.
   - ➢ $\forall$x: Roman(x) $\rightarrow$ LoyalTo(x,Caesar) $\vee$ Hate(x,Caesar)

Example of FOPL

6. Everyone is loyal to someone.

$\forall x: \exists y: LoyalTo(x,y)$

7. People only try to assassinate rulers they aren't loyal to.

$\forall x: \forall y: Person(x) \wedge Ruler(y) \wedge TryAssassinate(x,y) \rightarrow \neg LoyalTo(x,y)]$

8.Marcus tried to assassinate Caesar.

TryAssassinate(Marcus, Caesar)

9. All men are people.

$\forall x: Men(x) \rightarrow People(x)$

Prove this

- Given the above sentences, can we make a conclusion as follows:

“Marcus was not loyal to Caesar ?”

or:

¬loyalto(Marcus,Caesar)

Solution:

In order to prove the goal, we need to use the rules of inference to transform it into another goal that can in turn be transformed, and so on, until there are no unsatisfied goals remaining.

Solution

$\neg$ *loyalto*(*Marcus*, *Caesar*)

$\uparrow$ (7, substitution)

person(*Marcus*) $\wedge$
ruler(*Caesar*) $\wedge$
tryassassinate(*Marcus*, *Caesar*)

$\uparrow$ (4)

person(*Marcus*)
tryassassinate(*Marcus*, *Caesar*)

$\uparrow$ (8)

person(*Marcus*)

Fig. 5.2 *An Attempt to Prove ¬loyalto(Marcus,Caesar)*

Unification

- Any substitution that makes two or more expression equal is called a unifier for the expression.
- Two formulas unify if they can be made identical.
- A unification is a function that assigns bindings to variables.
- A binding is either a constant, a functional expression or another variable.

Unification process

- The basic idea of unification is very simple.
- To attempt to unify two literals, we first check if their initial predicate symbols are the same. If so, we can proceed.
- Otherwise, there is no way they can be unified, regardless of their arguments. For example, the two literals

tryassassinate (Marcus, Caesar)

hate(Marcus, Caesar)

cannot be unified.

If the predicate symbols match, then we must check the arguments, one pair at a time. If the first matches, we can continue with the second, and so on.

Unification Example

P(x, x)	P(A,A)	{x/A}
P(x, x)	P(A,B)	fail
P(x, y)	P(A,B)	{x/A, y/B}
P(x, y)	P(A,A)	fail
P(x, y)	P(A, z)	{x/A, y/z}
P(f(x),y)	P(f(A),B)	{x/A, y/B}
P(x, y)	P(f(x),B)	fail
P(x, y)	P(z, f(z))	{x/z, y/f(z)}

Unification algorithm

- Unify(L1, L2) returns a list representing the composition of the substitutions that were performed during the match.
- The empty list, NIL, indicates that a match was found without any substitutions.
- The list consisting of the single value FAIL indicates that the unification procedure failed.
- The final substitution produced by the unification process will be used by the resolution procedure.

Unification algorithm

```
Algorithm: Unify(L1, L2)
 I. If L1 or L2 are both variables or constants, then:
      (a) If L1 and L2 are identical, then return NIL.
      (b) Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return
          (L2/L1).
      (c) Else if L2 is a variable, then if L2 occurs in L1 then return {FAIL} , else return
          (L1/L2).
      (d) Else return {FAIL}.
 2. If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL}.
 3. If LI and L2 have a different number of arguments, then return {FAIL}.
 4. Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the
    substitutions used to unify L1 and L2.)
 5. For i ← 1 to number of arguments in L1 :
      (a) Call Unify with the ith argument of L1 and the ith argument of L2, putting result
          in S.
      (b) If S contains FAIL then return {FAIL}.
      (c) If S is not equal to NIL then:
            (i) Apply S to the remainder of both L1 and L2.
            (ii) $SUBST := \mathrm{APPEND}(S, SUBSET)$
 6. Return SUBST.
```

Solve

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say "fail":

a. R(A, x), R(y, z)

b. P(A, B, B), P(x, y, z)

c. Q(y, G(A, B)), Q(G(x,x), y)

d. Older(Father(y), y), Older(Father(x), John)

e. Knows(Father(y),y), Knows(x,x)

Solution

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say "fail":

a. R(A, x), R(y, z) y/A, x/z

b. P(A, B, B), P(x, y, z) x/A, y/B, z/B

c. Q(y, G(A, B)), Q(G(x,x), y) fail

d. Older(Father(y), y), Older(Father(x), John) x/y, y/John

e. Knows(Father(y),y), Knows(x,x) fail

Resolution Principle

- Given two clauses C_1 and C_2 with no variables in common, if there is a literal l_1, in C_1 and which is a complement of a literal l_2 in C_2 , both l_1 and l_2 are deleted and a disjuncted C is formed the remaining reduced clauses. The new clauses C is called the resolve of C_1 and C_2.
- Resolution is the process of generating these resolvents from the set of clauses.

Example: (~ PVQ) and (~Q V R)

- We can write

(~ PVQ) , (~Q V R)

~P V R

____

Refutation

- Resolution produces proofs 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.

Example of Resolution

- Consider the following clauses:-

  A : P V Q V R

  B: ~ P V Q V R

  C: ~Q V R

- Solution

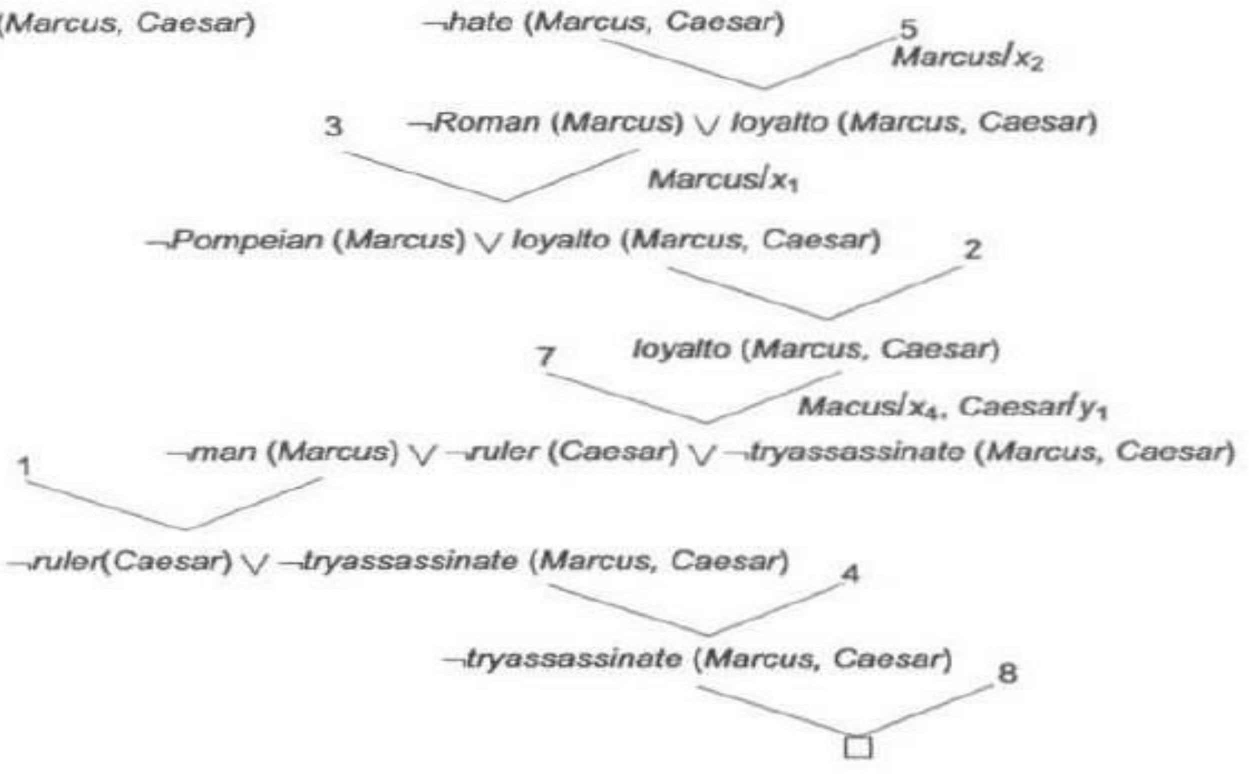
A: P V Q V R	(Given in the problem)
B : ~P V Q V R	(Given in the problem)
D : Q V R	(Resolvent of A and B)
C: ~Q V R	(Given in the problem)
E: R	(Resolvent of C and D)

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \lor Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \lor loyalto(x_2, Caesar) \lor hate(x_2, Caesar)$
6. $loyalto(x_3, fl(x_3))$
7. $\neg man(x_4) \lor \neg ruler(y_1) \lor \neg tryassassinate(x_4, y_1) \lor loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

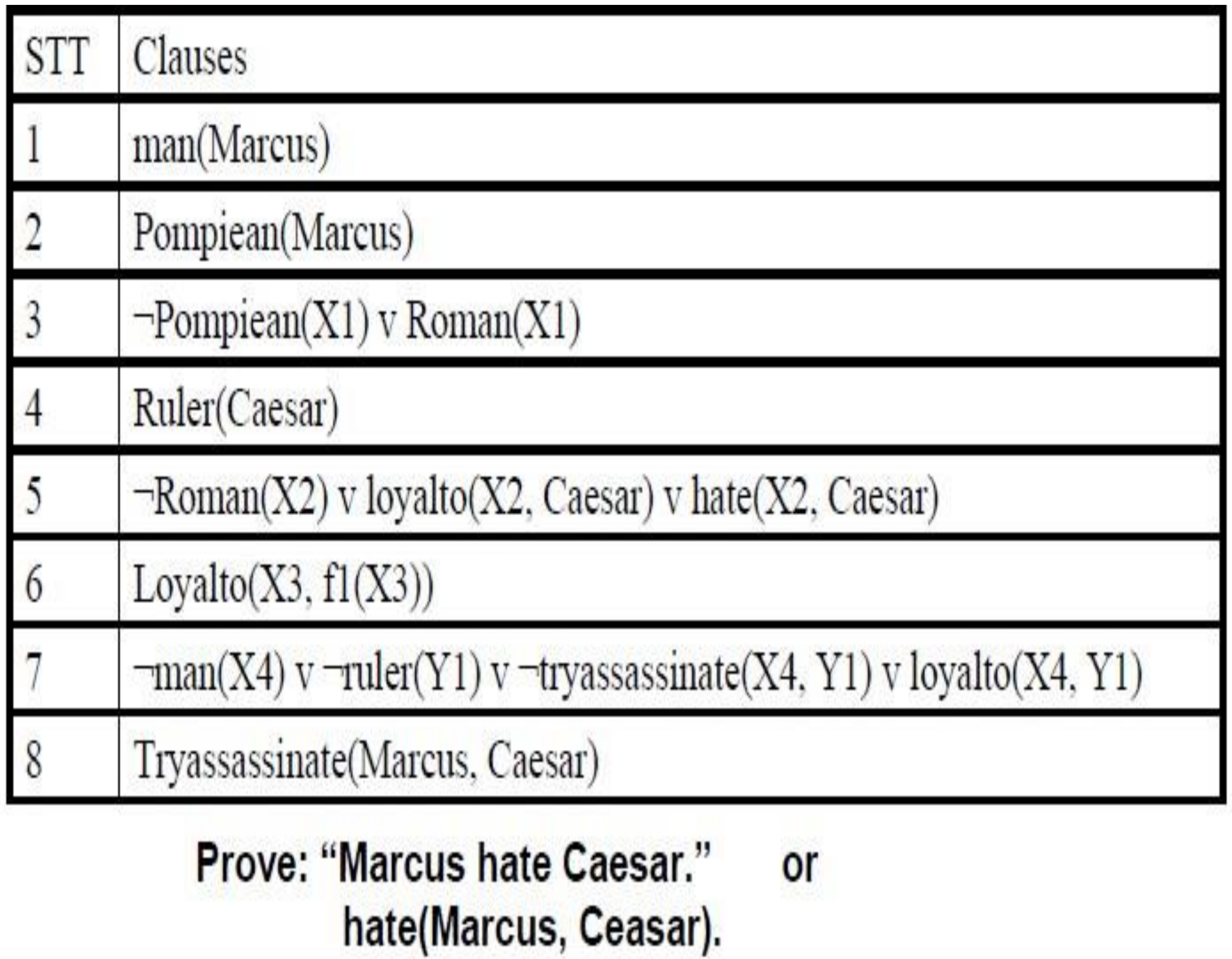
(a)

Prove: *hate* (*Marcus, Caesar*)



¬hate (Marcus, Caesar)
5
Marcus/x₂
¬Roman (Marcus) ∨ loyalto (Marcus, Caesar)
3
Marcus/x₁
¬Pompeian (Marcus) ∨ loyalto (Marcus, Caesar)
2
loyalto (Marcus, Caesar)
7
Macus/x₄, Caesar/y₁
¬man (Marcus) ∨ ¬ruler (Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
1
¬ruler(Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
4
¬tryassassinate (Marcus, Caesar)
8
□

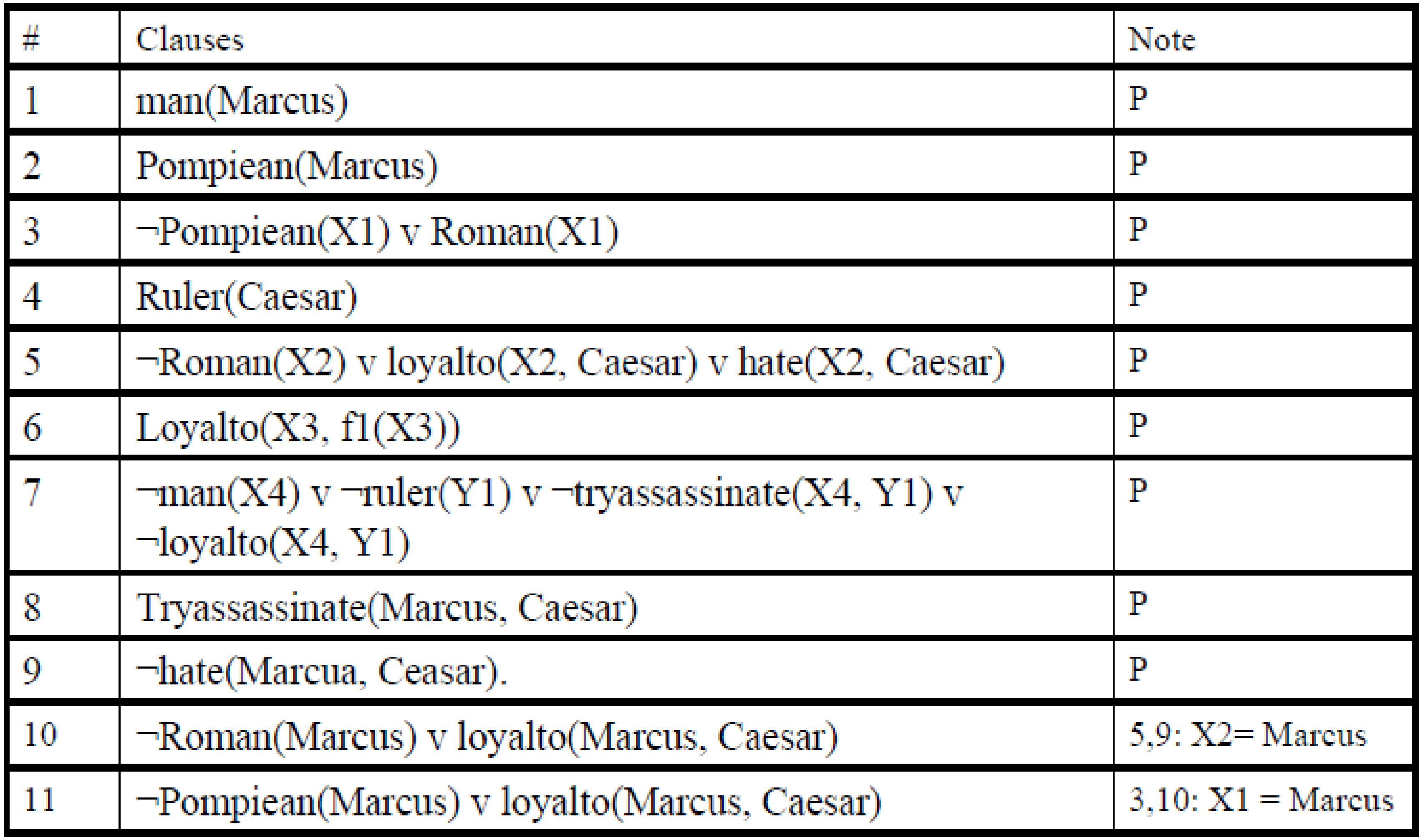
Example of resolution



STT	Clauses
1	man(Marcus)
2	Pompiean(Marcus)
3	¬Pompiean(X1) v Roman(X1)
4	Ruler(Caesar)
5	¬Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)
6	Loyalto(X3, f1(X3))
7	¬man(X4) v ¬ruler(Y1) v ¬tryassassinate(X4, Y1) v loyalto(X4, Y1)
8	Tryassassinate(Marcus, Caesar)

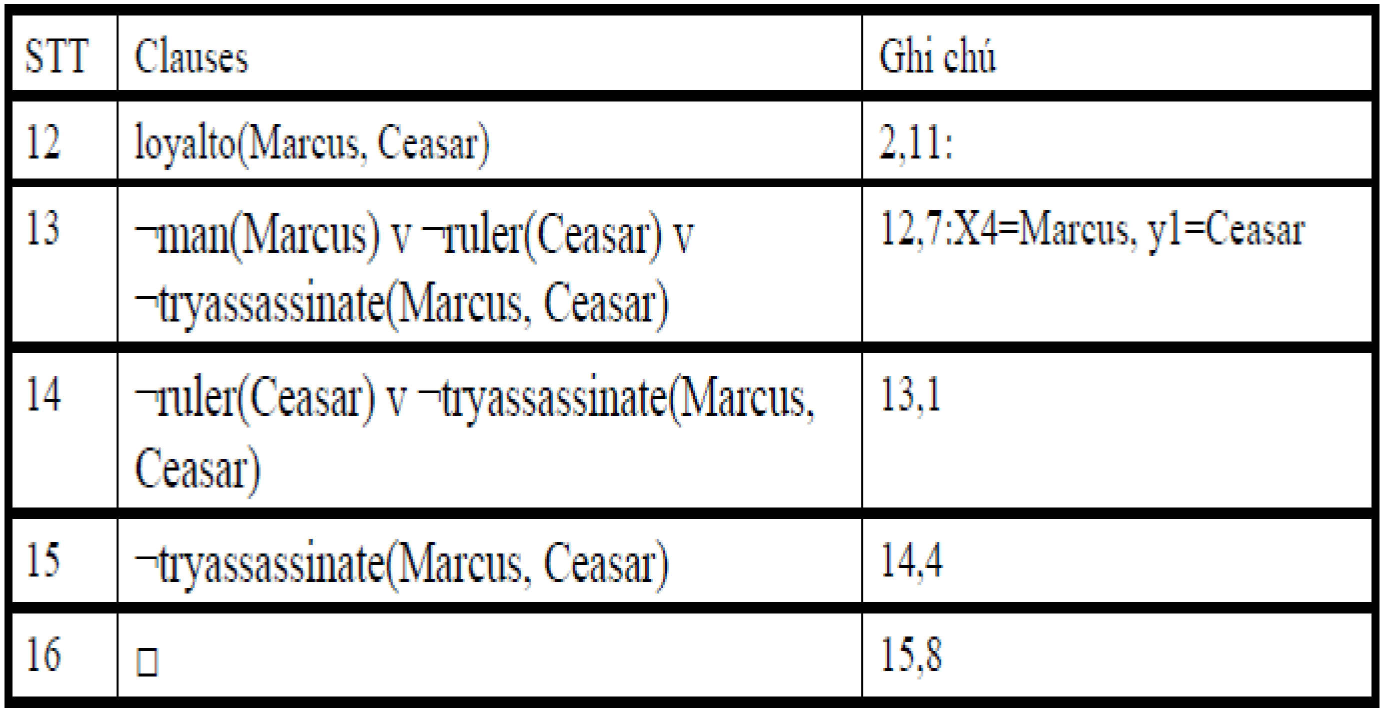
Prove: "Marcus hate Caesar." or hate(Marcus, Ceasar).

Solution



#	Clauses	Note
1	man(Marcus)	P
2	Pompiean(Marcus)	P
3	¬Pompiean(X1) v Roman(X1)	P
4	Ruler(Caesar)	P
5	¬Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)	P
6	Loyalto(X3, f1(X3))	P
7	¬man(X4) v ¬ruler(Y1) v ¬tryassassinate(X4, Y1) v ¬loyalto(X4, Y1)	P
8	Tryassassinate(Marcus, Caesar)	P
9	¬hate(Marcua, Ceasar).	P
10	¬Roman(Marcus) v loyalto(Marcus, Caesar)	5,9: X2= Marcus
11	¬Pompiean(Marcus) v loyalto(Marcus, Caesar)	3,10: X1 = Marcus

Solution



STT	Clauses	Ghi chú
12	loyalto(Marcus, Ceasar)	2,11:
13	¬man(Marcus) v ¬ruler(Ceasar) v ¬tryassassinate(Marcus, Ceasar)	12,7:X4=Marcus, y1=Ceasar
14	¬ruler(Ceasar) v ¬tryassassinate(Marcus, Ceasar)	13,1
15	¬tryassassinate(Marcus, Ceasar)	14,4
16	□	15,8

Example

Assume the following facts:

i. "Steve only likes easy courses.

ii. Science courses are hard.

iii. All the courses in Humanities Department are easy.

iv. HM101 is a course in Humanities".

Convert the above statements into appropriate wffs so that the resolution can be performed to answer the question. " what course would steve like?"

Solution

First we will convert it into FOPL (First order predicate logic)

i. “Steve only likes easy courses.

$\forall$x: easy(x) -> likes(steve,x)

ii. Science courses are hard.

$\forall$x: science(x) -> ~easy(x)

iii. All the courses in Humanities Department are easy.

$\forall$x: humanities(x) -> easy(x)

iv.HM101 is a course in Humanities”.

humanities(HM101)

The conclusion is encoded as likes(steve , x).

Solution continued….

- First we put our premises in the clause form and the negation of conclusion to our set of clauses (we use numbers in parentheses to number the clauses):

(1) ~easy(x) $\vee$ likes(steve,x)

(2) ~science(x) $\vee$ ~easy(x)

(3) ~humanities (x) $\vee$ easy(x)

(4) humanities(HM101)

(5) ~likes(steve,x)

Solution continued….

St	Clauses	Note
1	~easy(x) ∨ likes(steve,x)	P
2	~science(x) ∨ ~easy(x)	P
3	~humanities (x) ∨ easy(x)	P
4	humanities(HM101)	P
5	~likes(steve,x)	P
6	~easy(x)	1 & 5
7	~humanities (x)	3 & 6
8	NIL	4 & 7, x=HM101

$$\begin{tikzcd}
 & \neg\, likes(steve,x) \arrow[dr, dash] & & \neg\, easy(x) \lor likes(steve,x) \arrow[dl, dash] \\
\neg\, Humanities(x) \lor easy(x) \arrow[dr, dash] & & \neg\, easy(x) \arrow[dl, dash] & \leftarrow resolvent \\
 & \neg Humanities(x) \arrow[dr, dash] & & Humanities(HM101) \arrow[dl, dash] \\
 & & \square & HM101/X
\end{tikzcd}$$

Solution continued….

- A resolution proof may be obtained by the following sequence of resolutions
- (6) 1&5 yields resolvent ~easy(x).
- (7) 3&6 yields resolvent ~humanities (x).
- (8) 4&7 yields empty clause; the substitution x/HM101 is produced by the unification algorithm which says that the only wff of the form likes(steve,x) which follows from the premises is likes(steve, HM101). Thus, resolution gives us a way to find additional assumptions.

Example

Problem Statement:

1. Ravi likes all kind of food.
2. Apples and chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and is still alive
5. Rita eats everything that Ajay eats.

Prove by resolution that Ravi likes peanuts using resolution.

Solution

Step 1: Converting the given statements into Predicate/Propositional Logic

i. ∀x : food(x) → likes (Ravi, x)

ii. food (Apple) ^ food (chicken)

iii. ∀a : ∀b: eats (a, b) ^ killed (a) → food (b)

iv. eats (Ajay, Peanuts) ^ alive (Ajay)

v. ∀c : eats (Ajay, c) → eats (Rita, c)

vi. ∀d : alive(d) → ~killed (d)

vii. ∀e: ~killed(e) → alive(e)

Conclusion: likes (Ravi, Peanuts)

Solution continued…

Step 2: Convert into CNF

i. ~food(x) v likes (Ravi, x)

ii. Food (apple)

iii. Food (chicken)

iv. ~ eats (a, b) v killed (a) v food (b)

v. Eats (Ajay, Peanuts)

vi. Alive (Ajay)

vii. ~eats (Ajay, c) V eats (Rita, c)

viii. ~alive (d) v ~ killed (d)

ix.Killed (e) v alive (e)

Conclusion: likes (Ravi, Peanuts)

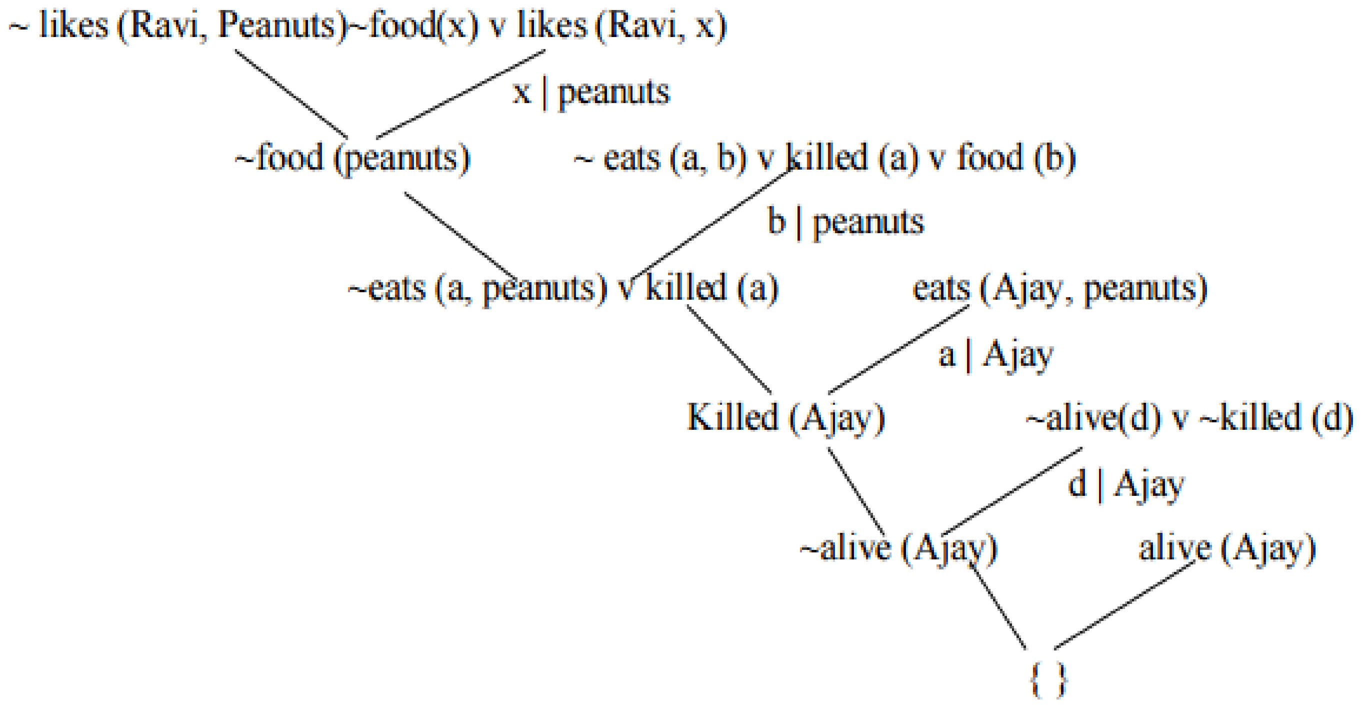
Solution continued…

Step 3: Negate the conclusion

~ likes (Ravi, Peanuts)

Step 4: Resolve using a resolution tree

Solution continued…



Hence we see that the negation of the conclusion has been proved as a complete contradiction with the given set of facts. Hence the negation is completely invalid or false or the assertion is completely valid or true.

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
RAINING $\rightarrow$ $\neg$SUNNY

Using Predicate Logic

Represent the facts in logic

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.

Using Predicate Logic

1. Marcus was a man.

   man(Marcus)

Using Predicate Logic

2. Marcus was a Pompeian.

   Pompeian(Marcus)

Using Predicate Logic

3. All Pompeians were Romans.

   $\forall x$: Pompeian(x) $\rightarrow$ Roman(x)

Using Predicate Logic

4. Caesar was a ruler.

   ruler(Caesar)

Using Predicate Logic

5. All Pompeians were either loyal to Caesar or hated him.

   inclusive-or

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

   exclusive-or

   $\forall x$: Roman(x) $\rightarrow$ (loyalto(x, Caesar) $\wedge$ $\neg$hate(x, Caesar)) $\vee$
   ($\neg$loyalto(x, Caesar) $\wedge$ hate(x, Caesar))

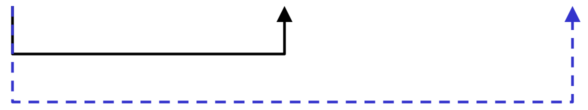
Using Predicate Logic

6. Every one is loyal to someone.

   $\forall x: \exists y: loyalto(x, y)$ $\exists y: \forall x: loyalto(x, y)$

Using Predicate Logic

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



$\forall x$: $\forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y)

$\rightarrow \neg$loyalto(x, y)

Using Predicate Logic

8. Marcus tried to assassinate Caesar.

   tryassassinate(Marcus, Caesar)

Using Predicate Logic

Was Marcus loyal to Caesar?

man(Marcus)

ruler(Caesar)

tryassassinate(Marcus, Caesar)

$\Downarrow$ $\quad\quad$ $\forall x$: man(x) $\rightarrow$ person(x)

$\neg$loyalto(Marcus, Caesar)

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).

Reasoning

1. Marcus was a Pompeian.
2. All Pompeians died when the volcano erupted in 79 A.D.
3. It is now 2008 A.D.

Is Marcus alive?

Reasoning

1. Marcus was a Pompeian.

   $Pompeian(Marcus)$

2. All Pompeians died when the volcano erupted in 79 A.D.

   $erupted(volcano, 79) \wedge \forall x: Pompeian(x) \rightarrow died(x, 79)$

3. It is now 2008 A.D.

   $now = 2008$

Reasoning

1. Marcus was a Pompeian.

   Pompeian(Marcus)

2. All Pompeians died when the volcano erupted in 79 A.D.

   $\text{erupted}(\text{volcano}, 79) \wedge \forall x: \text{Pompeian}(x) \rightarrow \text{died}(x, 79)$

3. It is now 2008 A.D.

   now = 2008

$\forall x: \forall t_1: \forall t_2: \text{died}(x, t_1) \wedge \text{greater-than}(t_2, t_1) \rightarrow \text{dead}(x, t_2)$

Resolution

- Resolution produces proofs by refutation(i.e. To prove statement)

The basic ideas

$$KB \models \alpha \quad \Leftrightarrow \quad KB \wedge \neg\alpha \models \text{false}$$

$$(\alpha \vee \neg\beta) \wedge (\gamma \vee \beta) \quad \Rightarrow \quad (\alpha \vee \gamma)$$

sound and complete

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.

   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
   $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
   $\neg \forall x\text{: } P \equiv \exists x\text{: } \neg P$
   $\neg \exists x\text{: } p \equiv \forall x\text{: } \neg P$
   $\neg\neg\, P \equiv P$

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

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

Conversion to Clause Form

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

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

5. Eliminate $\exists$ (Skolemization).

$\exists x: P(x) \equiv P(c)$	Skolem constant
$\forall x: \exists y\ P(x, y) \equiv \forall x: P(x, f(x))$	Skolem function

6. Drop $\forall$.

   $\forall x: P(x) \equiv P(x)$

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.

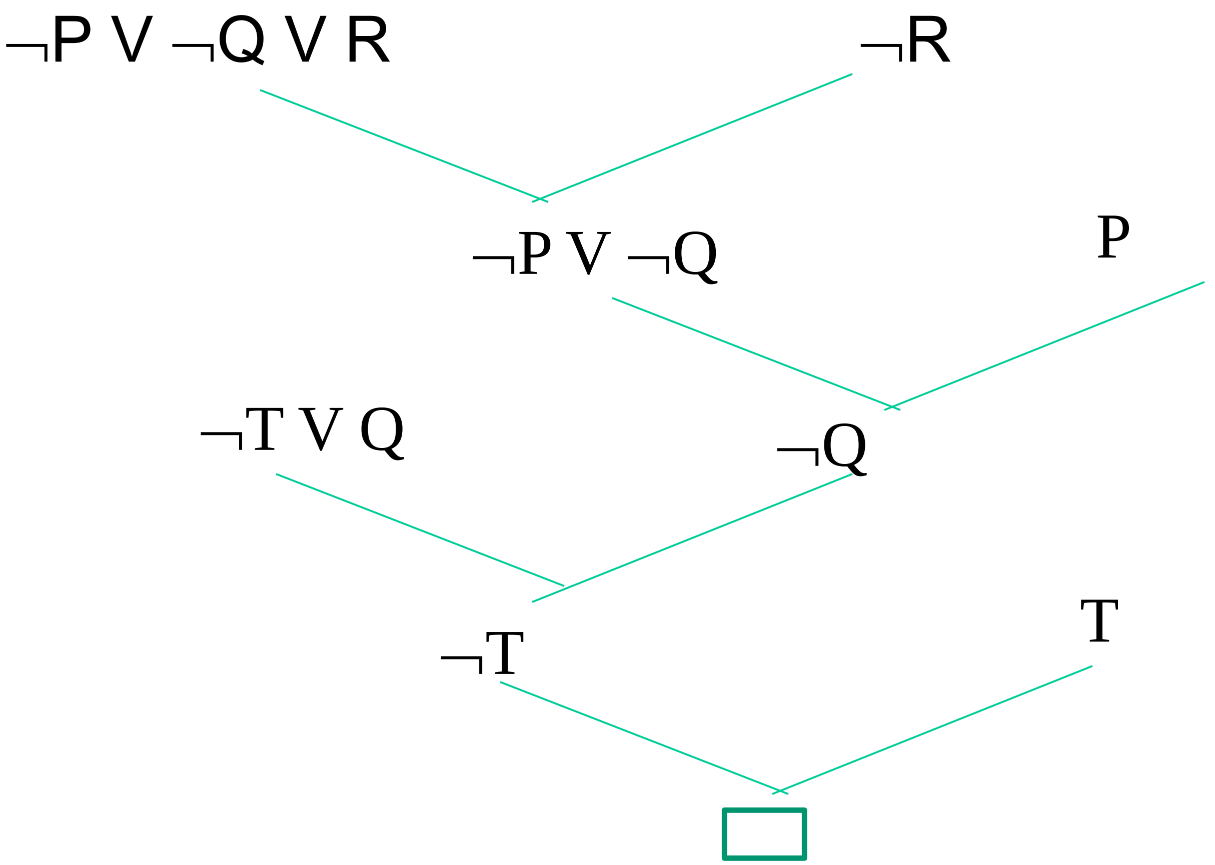
Resolution in Propositional Logic

1. Convert all the propositions of KB 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 ($\alpha \vee \neg P$) and ($\gamma \vee P$).
   b. Add the resolvent ($\alpha \vee \gamma$) to S.

Resolution in Propositional Logic

Example: given the Axioms Prove R

Given Axioms	converted to Clause form	
P	P	(1)
$(P \wedge Q) \rightarrow R$,	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$,	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	


¬P V ¬Q V R
¬R
¬P V ¬Q
P
¬T V Q
¬Q
¬T
T

Unification

UNIFY

- To unify P(x,x), P(x,y)   y/x substitute y for x
- (knows(John, x), knows(John, Jane)) = {Jane/x}

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

UNIFY(knows(John, x), knows(x, Elizabeth)) = FAIL

Algorithm :Unify(L1,L2)

1. If L1 or L2 are both variables or constants, then

   (a) If L1 and L2 are identical, then return NIL

   (b) Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return (L2/L1).

   (C) Else if L2 is a variable then if L2 occurs in L1 then return {FAIL},else return (L1/L2).

   (d) Else return {FAIL}.

2. If the initial predicate symbols in L1 and are not identical, then return {FAIL}.

```
3.  If L1 and L2 have a different number of arguments, then return
    {FAIL}.
4.  Set  SUBST to NIL.
5.  For i←1 to number of arguments in L1
    (a) Call Unify with ith argument of L1 and the ith        argument
    of L2 ,put result in S
    (b) If  S contain FAIL then return {FAIL}
    (c) If S is not equal to NIL then
         i  Apply S to remainder of both L1 and L2
         ii  SUBST:= APPEND(S,SUBST)
6. Return SUBST.
```

Resolution in predicate Logic

- Resolve two clauses
  1. man(Marcus)
  2. ¬man(x1) V mortal(x1)

  two clauses can be unified substitute Marcus/x1

$$\begin{tikzcd}
\text{man(Marcus)} \arrow[dr, dash] & & \neg\text{man(Marcus)} \arrow[dl, dash] & \leftarrow \text{parent clause} \\
& \text{mortal(Marcus)} & & \leftarrow \text{Resolvent}
\end{tikzcd}$$

Resolution Algorithm

- F is set of given statements
- We want to prove statement P

- Convert All the statements of F to Clause Form
- Negate P and Convert it into Clause Form
- Repeat Until either Contradiction is found or no progress can be made

  (a) Select two clauses call them parent clauses

  (b) Resolve them together

  if pair of literals T1 and ¬T2, one parent clause contain T1 and other contain ¬T2

  T1 and T2 are Unifiable

  (C) If the resolvent is empty then contradiction has been found ,other repeat the procedure

Example

Q. Assume Following facts and give the Answer of the Question

Is Marcus hate Caesar? Using Resolution

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.

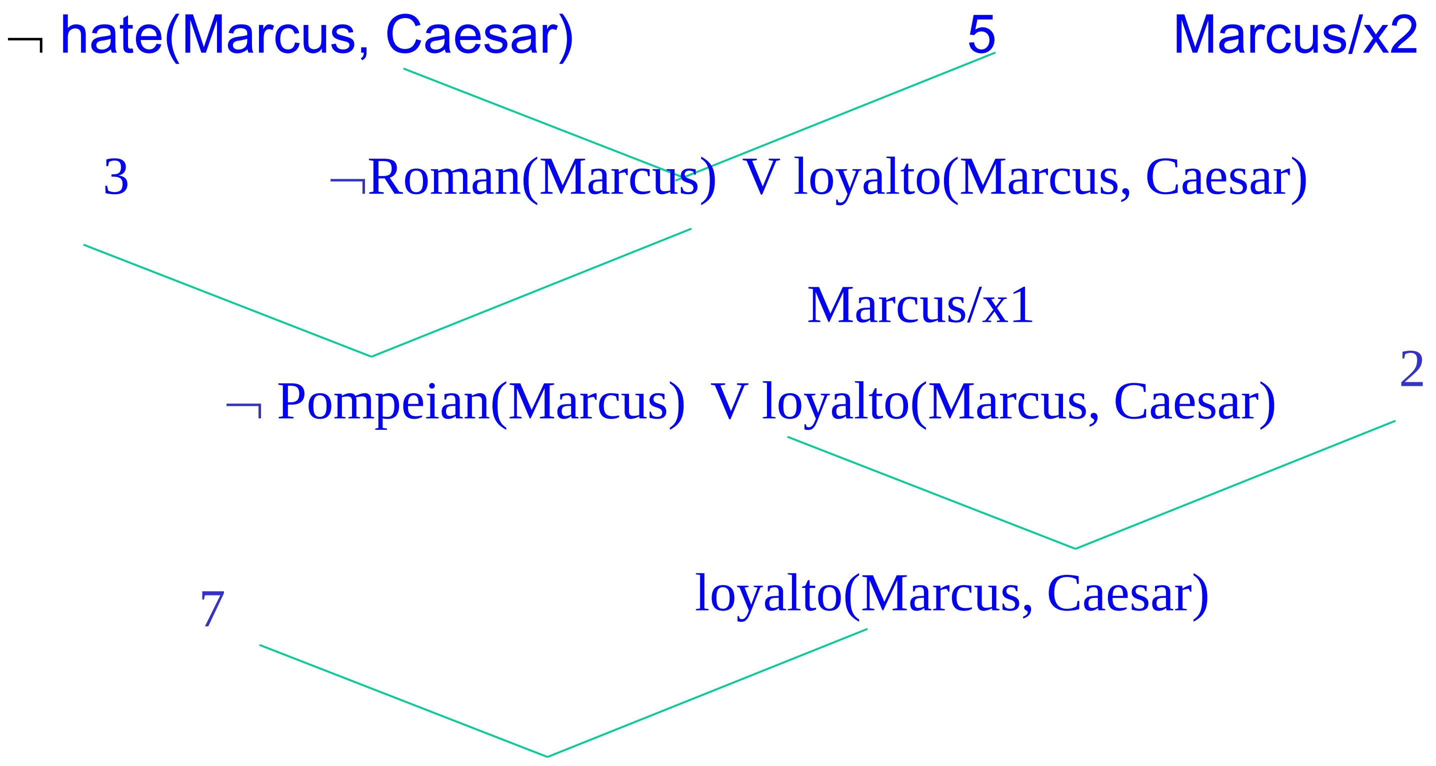
Example

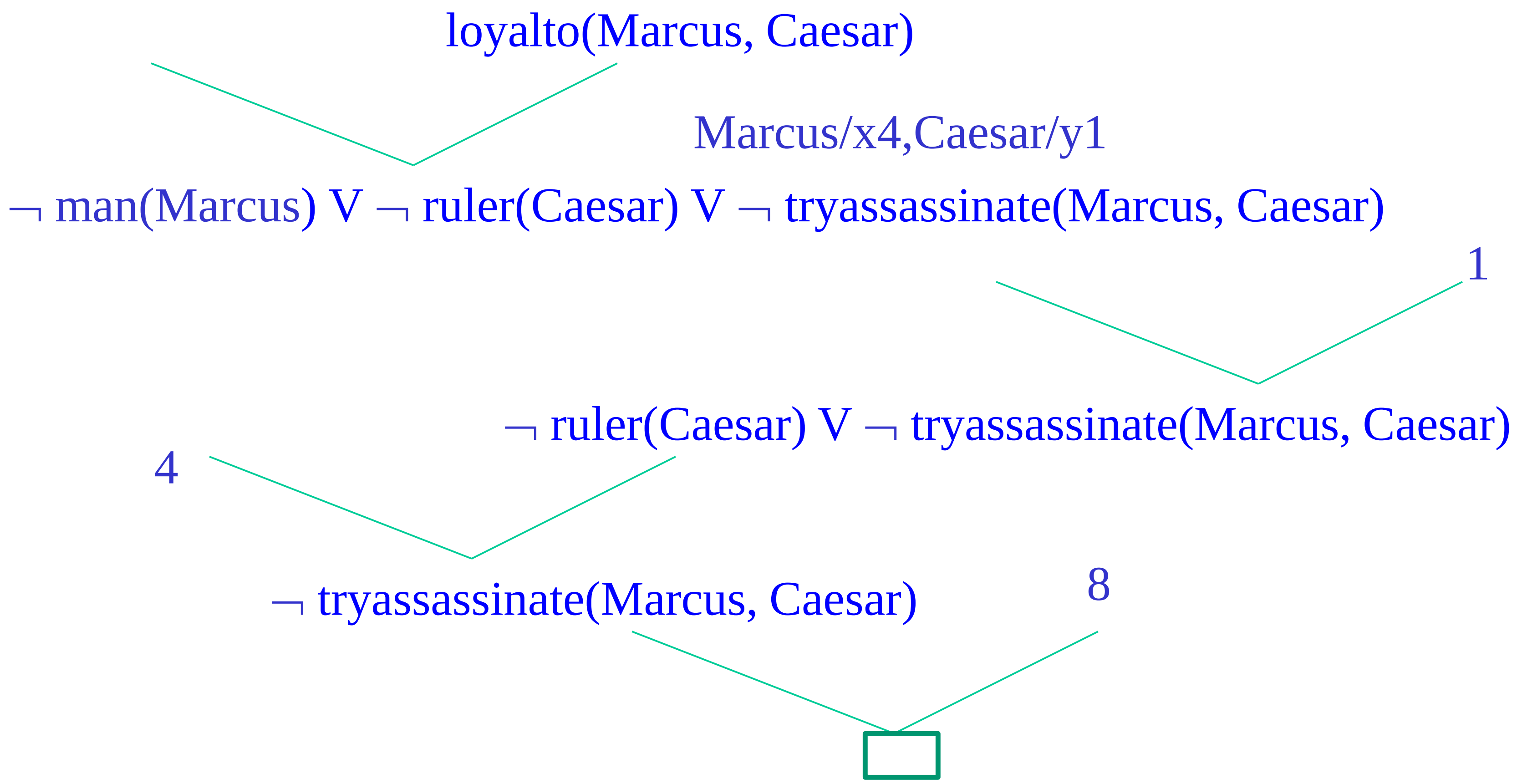
Axioms in Clause Form

1. Man(Marcus).
2. Pompeian(Marcus).
3. ¬ Pompeian(x1) V Roman(x1).
4. ruler(Caesar).
5. ¬Roman(x2) V loyalto(x2, Caesar) ∨ hate(x2, Caesar).
6. loyalto(x3, f1(x3)).
7. ¬ man(x4) ∧ ¬ ruler(y1) ∧ ¬ tryassassinate(x4, y1) V ¬ loyalto(x4, y1).
8. tryassassinate(Marcus, Caesar).

Example

Prove: hate(Marcus, Caesar)


¬ hate(Marcus, Caesar)
5
Marcus/x2
3
¬Roman(Marcus) V loyalto(Marcus, Caesar)
Marcus/x1
2
¬ Pompeian(Marcus) V loyalto(Marcus, Caesar)
7
loyalto(Marcus, Caesar)


loyalto(Marcus, Caesar)
Marcus/x4,Caesar/y1
¬ man(Marcus) V ¬ ruler(Caesar) V ¬ tryassassinate(Marcus, Caesar)
1
¬ ruler(Caesar) V ¬ tryassassinate(Marcus, Caesar)
4
¬ tryassassinate(Marcus, Caesar)
8

A Resolution Proof

Use resolution to prove marcus is not alive now

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991 A.D.

Question Answering

1. When did Marcus die?
2. Whom did Marcus hate?
3. Who tried to assassinate a ruler?
4. What happen in 79 A.D.?.
5. Did Marcus hate everyone?

Use Resolution to prove john likes Peanuts

1 John likes all kinds of food.

2 Apples are food.

3 Banana is food.

4 Anything anyone eats and isn't killed by is food.

5 Bill eats peanuts and is still alive.

6 Sue eats everything Bill eats.

Solution :

covert sentences into logic

1. $\forall x$:food(x) $\rightarrow$ likes(John , x)

2. food(apples)

3. food(banana)

4. $\forall x \forall y$ eats(y , x) $\wedge$ alive(y) $\rightarrow$ food(x)

5. $\forall x$ eats(Bill , x) $\wedge$ alive(Bill)

6. $\forall x$ eats(Bill , x) $\rightarrow$ eats(Sue , x)

7. $\neg$ likes(John , peanuts)

Convert logic form into clause form

1. $\neg$ food(x) $\vee$ likes(John , x)
2. food(apples)
3. food(banana)
4. $\neg$ eats(y , z) $\vee$ $\neg$ alive(y) $\vee$ food(z)
5. eats(Bill , peanuts)
6. alive(Bill)
7. $\neg$ eats(bill , w) $\vee$ eats(Sue , w)
8. $\neg$ likes(John , peanuts)

- Use resolution to answer the question, “What food does Sue eat?”
- What food does Sue eat?”

  Eats(Sue,X),

  Negate Query $\neg$ Eats(Sue, X)

$$\begin{tikzcd}
\neg \text{Eats(Sue, X)} \arrow[dr, no head] & & \text{Eats(Bill,peanuts)} \arrow[dl, no head, "\text{P eanuts/x}"] \\
& \neg \text{Eats(Sue,peanuts)} &
\end{tikzcd}$$

- Prove that john likes peanuts using backward chaining
- Forward chaining : attempt to from start state
- Backward chaining : attempt to prove from goal state

Use backward chaining to prove that John likes peanuts.

- To prove Likes(John, Peanuts),
- we just need to prove Food(Peanuts), according to sentence 1.
- By sentence 4, we just need to prove
- Eats(y, Peanuts) ∧ ¬KilledBy(y, Peanuts) **V food(Penuts)**.

the following statements are assumed to be true:

- Steve only likes easy courses.
- Science courses are hard.
- All the courses in the basketweaving department are easy.
- BK301 is a basketweaving course.

Using Resolution find the Answer

What course would Steve like?

Convert sentences into predicate logic

- $\forall \mathbf{x}$ easy(x) -> likes(steve,x)
- $\forall \mathbf{x}$ science(x) -> $\neg$ easy(x)
- $\forall \mathbf{x}$ basketweaving(x) -> easy(x)
- basketweaving(BK301)

- First convert it into clause form
- Negation of conclusion to our set of clauses

(1) $\neg$ easy(x) $\vee$ likes(steve,x)

(2) $\neg$ science(x) $\vee$ $\neg$ easy(x)

(3) $\neg$ science(x) $\vee$ $\neg$ easy(x)

(4) $\neg$ basketweaving(x) $\vee$ easy(x)

(5) basketweaving(BK301)

(6) $\neg$ likes(steve,x)

$$\begin{tikzcd}
 & \neg\,\text{likes(steve,x)} \arrow[dr, dash] & & \neg\,\text{easy(x)} \vee \text{likes(steve,x)} \arrow[dl, dash] \\
\neg\,\text{basketweaving(x)} \vee \text{easy(x)} \arrow[dr, dash] & & \neg\,\text{easy(x)} \arrow[dl, dash] & \leftarrow \text{resolvent} \\
 & \neg\,\text{basketweaving(x)} \arrow[dr, dash] & & \text{basketweaving(BK301)} \arrow[dl, dash] \\
 & & \square & \text{BK301/X}
\end{tikzcd}$$

Forward Chaining

Forward Chaining the Inference Engine goes through all the facts, conditions and derivations before deducing the outcome i.e When based on available data a decision is taken then the process is called as Forwarding chaining, It works from an initial state and reaches to the goal(final decision).

Example:

A

A -> B

B

____

He is running.

If he is running, he sweats.

He is sweating.

Example : Forward Chaining

Example:	Forward Chaining	
Rule R1	IF hot AND smoky	THEN ADD fire
Rule R2	IF alarm_beeps	THEN ADD smoky
Rule R3	If fire	THEN ADD switch_on_sprinklers
Fact F1	alarm_beeps	[Given]
Fact F2	hot	[Given]
Fact F3	**smoky**	**[from F1 by R2]**
Fact F4	**fire**	**[from F2, F3 by R1]**
Fact F5	**switch_on_sprinklers**	**[from F4 by R3]**

Fact F3,F4 and F5 are derived from the given facts and rules.

Backward Chaining

In this, the inference system knows the final decision or goal, this system starts from the goal and works backwards to determine what facts must be asserted so that the goal can be achieved, i.e it works from goal(final decision) and reaches the initial state.

Example:

B

A -> B

A

He is sweating.

If he is running, he sweats.

He is running.

Example : Backward Chaining

Example:	Forward Chaining	
Rule R1	IF hot AND smoky	THEN ADD fire
Rule R2	IF alarm_beeps	THEN ADD smoky
Rule R3	If fire	THEN ADD switch_on_sprinklers
Fact F1	alarm_beeps	[Given]
Fact F2	hot	[Given]
Goal:	**Should I switch sprinkles on?**	

Using the given goal the facts are derived and answer is provided.

	Forward Chaining	Backward Chaining
1.	When based on available data a decision is taken then the process is called as Forward chaining.	Backward chaining starts from the goal and works backward to determine what facts must be asserted so that the goal can be achieved.
2.	Forward chaining is known as data-driven technique because we reaches to the goal using the available data.	Backward chaining is known as goal-driven technique because we start from the goal and reaches the initial state in order to extract the facts.
3.	It is a bottom-up approach.	It is a top-down approach.
4.	It applies the Breadth-First Strategy.	It applies the Depth-First Strategy.
5.	Its goal is to get the conclusion.	Its goal is to get the possible facts or the required data.
6.	Slow as it has to use all the rules.	Fast as it has to use only a few rules.
7.	It operates in forward direction i.e it works from initial state to final decision.	It operates in backward direction i.e it works from goal to reach initial state.
8.	Forward chaining is used for the planning, monitoring, control, and interpretation application.	It is used in automated inference engines, theorem proofs, proof assistants and other artificial intelligence applications.