Consider the following set of sentences:

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 are not loyal to.

8. Marcus tried to assassinate Caesar.

Wff in FOPL

1. man(Marcus)
2. Pompeian(Marcus)
3. ∀x Pompeian(x) 🡪 Roman(x)
4. Ruler(Ceaser)
5. ∀x : Roman(x) 🡪 loyalto(x,Ceaser) V hate(x, Ceaser)
6. ∀x : 🡪 ∃y :loyalto(x,y)
7. ∀x :∀y: Person(x) ∧ ruler(y) ∧ trytoAssasinate(x,y) 🡪 ~loyalto(x,y)
8. trytoAssasinate(Marcus,Ceaser)

9) All men are people ----∀x man(x) 🡪 person(x)

Question : **Was Marcus loyal to Caesar?**

~loyalto(Marcus, Ceaser) P∧T=P PVT=T P∧F=F PVF=P

7 x/Marcus, y/Ceaser

Person(Marcus) ∧ ruler(Ceaser) ∧ trytoAssasinate(Marcus,Ceaser)

8

Person(Marcus) ∧ ruler(Ceaser)

4

Person(Marcus)

9

Man(Marcus)

1

Nil

**2)** Translate following sentences into WFFs in predicate logic.

i) John likes all kinds of food.

ii) Apples are food. Chicken is food.

iii) Anything anyone eats and is not killed by is food.

iv) Bill eats peanuts and is still alive.

v) Sue eats everything Bills eats.

Prove that " John like peanuts " using backward chaining

**3)**

Ramu is a soldier.

Ramu is a resident of Madras.

Madras is in India. All Indian soldiers know Hindi.

Convert them into predicate form and resolve to answer the question “Does Ramu know Hindi?” (What additional information is needed to answer the question?)

**REPRESENTING INSTANCE AND *ISA* RELATIONSHIPS**

1. Instance(Marcus,man)
2. Instance(Marcus,Pompeian)
3. ∀x instace(x,Pompeian) 🡪 instance(x,Roman)
4. Instance(Ceaser,ruler)
5. ∀x : instance(x,Roman) 🡪 loyalto(x,Ceaser) V hate(x, Ceaser)

***ISA***

*3)isa(Pompeian,Roman)*

***COMPUTABLE FUNCTIONS AND PREDICATES:***

*gt(1,0), gt(2,1) gt(3,2) …………………………..*

*gt(2+3,1)*

*gt(a-b,0)*

*Consider the stmts*

1 . Marcus was a man.

2. Marcus was a Pompeian.

*3. Marcus was born in 40.AD*

*4. All men are mmortal.*

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

*6. No mortal lives longer than 150 years.*

*7. It is now 2021.*

*Represent these fact in FOPL*

1. man(Marcus)
2. Pompeian(Marcus)
3. *Born(Marcus, 40)*
4. ∀x :man(x) 🡪 mortal(x)
5. 1) Erupted(Volcano, 79) ∧

2 ) ∀x :[Pompeian(x) 🡪 died(x,79)]

1. ∀x : ∀t1 : ∀t2: mortal(x) ∧ born (x,t1) ∧ gt(t2-t1,150) 🡪 died(x, t2)
2. Now=2021

*Suppose We want “ Is Marcus alive? “*

1. ∀x : ∀t : [alive(x,t) 🡪 ~dead(x,t)] V[~dead(x,t) 🡪 alive(x,t) ]
2. ∀x : ∀t1: ∀t2 : died(x,t1) ∧ gt(t2,t1) 🡪 dead*(x,t2)*

~alive(Marcus,now)

8,7

dead(Marcus,now)

9 x/Marcus, t2/now

died(x,t1) ∧ gt(now,t1)

5) 2 t1/79, , substituion

Pompeian(Marcus) ∧ gt(now,79)

2)

gt(now,79)

7, substituion

gt(2021,79)

nil

**Resolution:**

**Normal forms**

Clause form (CNF)

***Algorithm: Convert to Clause Form***

1. ***Eliminate implication*** 🡪 (using a🡪 b = ~aVb)
2. Reduce scope of ~ ~(aVb)= ~a ∧ ~b , ~(a∧b)= ~a V ~b, ~(~a)=a,

~∀x :P(x)= ∃x:~P(x), ~∃x :P(x)= ∀x:~P(x)

1. Standardize variables
2. Move all quantifiers to left
3. Eliminate Existential quantifier.
4. Drop the prefix
5. Convert the matrix in conjunction of disjunction.
6. Create separate clauses
7. Standardize variables

Consider

All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy.

∀x : [Roman(x) ∧ know(x,Marcus)] 🡪 [hate(x, Ceaser) V ( ∀y : ∃z : hate(y,z)🡪 thinkcrazy(x,y))]

Applying Clause form

1. Eliminate 🡪 (using a🡪 b = ~aVb)

∀x : ~ [Roman(x) ∧ know(x,Marcus)] V [hate(x, Ceaser) V ( ∀y : ~( ∃z : hate(y,z) ) V hinkcrazy(x,y))]

1. Reduce ~ scope by ~(a∧b)= ~a V ~b and ~∃x :P(x)= ∀x:~P(x)

∀x : ~ Roman(x) V ~ know(x,Marcus)] V [hate(x, Ceaser) V ( ∀y : ∀z : ~ hate(y,z) ) V hinkcrazy(x,y))]

1. Standardize variables ------------------not required
2. Move all quantifiers to left

∀x : ∀y : ∀z : ~ Roman(x) V ~ know(x,Marcus)] V [hate(x, Ceaser) V ( ~ hate(y,z) ) V thinkcrazy(x,y))]

**PNX (Prenex Normal Form)**

1. Eliminate Existential quantifier. (SKolem Constant, SKolem functions)

∃z :Ruler(z) -------------- > Ruler (S1)

∀x : ∃z :father-of(z,x) ------------------- > ∀x : father-of(S2(x),x)

1. Drop the prefix

: ~ Roman(x) V ~ know(x,Marcus)] V [hate(x, Ceaser) V ( ~ hate(y,z) ) V thinkcrazy(x,y))]

1. Convert the matrix in conjunction of disjunction

(aVb) ∧ c

(a ∧b) V c = (aVc) ∧ (bVc)

1. Create separate clauses
   1. aVc
   2. bVc
2. Standardize variables

~ Roman(x) V ~ know(x,Marcus)] V hate(x, Ceaser) V ( ~ hate(y,z) ) V thinkcrazy(x,y))

**Resolution**

Winter V Summer and ~Winter V cold

Summer V cold (Resolvent)

Pvq ~qVr ------------------- > PVr

Algorithm : Propositional Resolution

1. ***Convert to Clause Form***
2. ***Negate P and convert it in clause form***
3. ***Repeat until contradiction or no progress***
   1. ***Select any 2 clause***
   2. ***Resolve them L ~L remaining is resolvent***
   3. ***If resolvent is empty then contradiction. Else add it to set of clauses***

Given

1. P
2. (P ∧ Q) 🡪 R
3. (S V T) 🡪 Q
4. T

Prove ---R

1. P
2. ~ P V ~Q V R
3. (~S ∧ ~T) V Q--------- (~S VQ) ∧ (~TVQ)

3.1) (~S VQ)

3.2) (~TVQ)

4) T

R ~ P V ~Q V R

~ P V ~Q 1) P

~Q 3.2 ) ~TVQ

` ~T 4)T

Nil

**The Unification Algorithm**

In propositional logic, it is easy to determine that two literals cannot both be true at the same time. Simply look for *L* and *¬ L* in predicate logic, this matching process is more complicated since the arguments of the predicates must be considered. For example, *man(John)* and *¬ man(John)* is a contradiction, while *man(John)* and *¬ man(Spot)* is not.

Thus, in order to determine contradictions, we need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical. There is a straightforward recursive procedure, called the *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 *i*th argument of *L1 and the i*th 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: =* APPEND(S*, SUBST*).

6. Return *SUBST.*

***Algorithm: Resolution***

1. Convert all the statements of *F* to clause form.

2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in 1.

3. Repeat until either a contradiction is found , no progress can be made, or a predetermined amount of effort has been expended.

(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals *T1* and ¬ *T2* such that one of the parent clauses contains *T2* and the other contains *T1* and if *T1* and *T2* are unifiable, then neither *T1* nor *T2* should appear in the resolvent. We call *T1* and *T2 Complementary literals.* Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Wff in FOPL

1. man(Marcus)
2. Pompeian(Marcus)
3. ∀x Pompeian(x) 🡪 Roman(x)
4. Ruler(Ceaser)
5. ∀x : Roman(x) 🡪 loyalto(x,Ceaser) V hate(x, Ceaser)
6. ∀x : 🡪 ∃y :loyalto(x,y)
7. ∀x :∀y: Person(x) ∧ ruler(y) ∧ trytoAssasinate(x,y) 🡪 ~loyalto(x,y)
8. trytoAssasinate(Marcus,Ceaser)
9. ∀x man(x) 🡪 person(x)

Prove by resolution that Marcus hate Ceasor

Resolution

~Roman(x1) V (loyalto(x1,Ceaser) V hate(x1, Ceaser))

~Person(x4) V ~ruler(y1) V ~ trytoAssasinate(x4,y1) ~~loyalto(x4,y1)

Step 1: Clause form

1. man(Marcus)
2. Pompeian(Marcus)
3. ~Pompeian(x1) V Roman(x1)
4. Ruler(Ceaser)
5. ~Roman(x2) V loyalto(x2,Ceaser) V hate(x2, Ceaser)
6. loyalto(x3,F(x3))
7. ~Person(x4) V ~ruler(y1) V ~ trytoAssasinate(x4,y1) V~loyalto(x4,y1)
8. trytoAssasinate(Marcus,Ceaser)
9. ~man(x5) Vpersonn(x5)

Step 2: Negate p and conver

~hate(Marcus,Ceasor)

Step 3.

~hate(Marcus,Ceasor) 5

Marcus/x2

~Roman(Marcus) V loyalto(Marcus ,Ceaser) 3

Marcus/x1

2 ~Pompeian(Marcus) ) V loyalto(Marcus ,Ceaser)

loyalto(Marcus ,Ceaser) 7

Marcus/x4, Ceaser/y1

~Person(Marcus) V ~ruler(Ceaser) V ~ trytoAssasinate(Marcus,Ceaser) 4

8 ~Person(Marcus) V ~ trytoAssasinate(Marcus,Ceaser)

~Person(Marcus) 9

Predicate Logic

1. Every student likes all AI modules .

∀𝑥∀𝑦 Student(x) ⋀ Module(y,AI) 🡪 Likes(x,y)

1. Every student likes an AI modules .

∀𝑥 ∃y Student(x) ⋀ Module(y,AI) 🡪 Likes(x,y)

1. All purple mushrooms are poisonous.

∀x Mushroom(x) ⋀ Purple(x) 🡪 Poisonous(x,y)

1. There are exactly two mangoes.

∃x ∃y Mango(x) ⋀ Mango(y) ⋀ ~(x=y) ⋀ ∀z Mango(z) 🡪 (x=z) V (y=z)

1. Every cszhild loves an Ice-cream.
2. Any student who does not pass does not play.
3. Any student who does not study does not pass.
4. **Only one student failed in AI.**
5. Brothers are siblings
6. You can fool all of the people some of the time

Nil