Proposition Logic

BY

DR. ANUPAM GHOSH

DATE: 20.11.2024

Logic is a <u>truth-preserving</u> <u>system</u> of <u>inference</u>

Truth-preserving: If the initial statements are true, the inferred statements will be true

System: a set of mechanistic transformations, based on syntax alone

Inference: the process of deriving (inferring) new statements from old statements

Logic: Other definitions

Any ‘formal system’ can be considered a logic if it has:

- – a well-defined syntax;
- – a well-defined semantics; and
- – a well-defined proof-theory
- The syntax of a logic defines the syntactically acceptable objects of the language, which are properly called well-formed formulae (wff). (We shall just call them formulae.)
- The semantics of a logic associate each formula with a meaning.
- The proof theory is concerned with manipulating formulae according to certain rules.

Proposition Logic

- The simplest, and most abstract logic we can study is called propositional logic.
- Definition: A proposition is a statement that can be either true or false; it must be one or the other, and it cannot be both
- It is possible to determine whether any given statement is a proposition by prefixing it with:

  It is true that . . .

  and seeing whether the result makes grammatical sense.
- A number of connectives which will allow us to build up complex propositions

Basic Operations

The NOT operation: ¬

The AND operation: ∧

The OR operation: ∨

The "implication" operation: →

The "equivalence" operation: ↔ or ≡

NOT Operation

- In plain text, we can describe the operation to be the opposite of what the original value is.

Example: Let p = "cat"
Then ¬p will be "not a cat"

Example: Let p = "tired"
Then ¬p = "not tired"

AND Operation

- In plain text,

Example: Let p = “tired” and q = “hungry”
Then $p \wedge q$ = “tired and hungry”
Also, $\neg p \wedge q$ = “not tired and hungry”
Or, $p \wedge \neg q$ = “tired and not hungry”

OR Operation

In plain text,

Example: Let p = “tired” and q = “hungry”
If “tired and not hungry”, $p \vee q$ is “true”
If “not tired and not hungry”, $p \vee q$ is “false”

“Implication” Operation

- For implication, it is slightly more complicated and requires two variables. It will return “true” if the initial condition is false, regardless of the value of the second variable, and when both variables are true. (See the truth table below)
- Denoted by the symbol “$\rightarrow$”
- Example: $p \rightarrow q$

Truth Table:

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

"Implication" Operation

- $p \rightarrow q$ is also equivalent (the same as) $\neg p \vee q$
- We can verify this using truth tables:

p	q	¬p V q
0	0	1
0	1	1
1	0	0
1	1	1

Equivalence

- Equivalence requires two variables and will return "true" if both variables have the same value
- Often denoted by the symbol "$\leftrightarrow$" or "$\equiv$"
- Example: $p \leftrightarrow q$ or $p \equiv q$

Truth Table:

p	q	p ↔ q
0	0	1
0	1	0
1	0	0
1	1	1

Biconditional $\leftrightarrow$

- The biconditional of p and q: $p \leftrightarrow q \triangleq (p \to q) \land (q \to p)$
  - True only when p and q have identical truth values
- *If and only if (iff)*

p q	$p \to q$	$q \to p$	$p \leftrightarrow q$
T T	T	T	T
T F	F	T	F
F T	T	F	F
F F	T	T	T

Known operators:

$\land$ conjunction (and),

$\lor$ disjunction (or),

$\neg$ negation,

$\to$ conditional (if then)

Operator Precedence

- From high to low: ( ), $\neg$, $\land\lor$, $\to$, $\leftrightarrow$
- When equal priority instances of *binary connectives* are not separated by (), the *leftmost one has precedence*. E.g. $p \to q \to r \equiv (p \to q) \to r$
- When instances of $\neg$ are not separated by (), the *rightmost one has precedence*:

  E.g. $\neg\neg\neg p \equiv \neg(\neg(\neg p))$

Tautology

- Tautology is very similar to logical equivalence
- When all values are “true” that is a tautology

Example: $p \equiv q$ if and only if $p \leftrightarrow q$ is a tautology

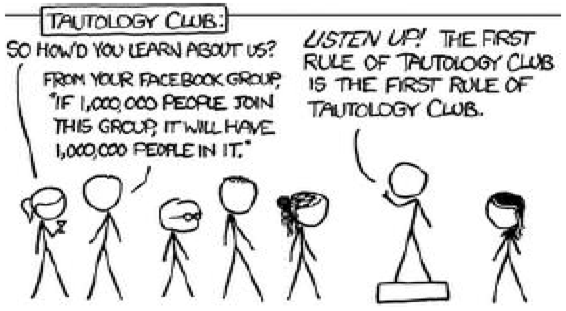
Example: $p \equiv \neg\neg p$ is a tautology

An expression that always gives a true value is called a tautology .

Example: $p \vee (\neg p) \equiv T$

p	¬p	p∨¬p
T	F	T
F	T	T

Always true!



Example: Show that $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ ← statement

p	q	r	q V r	p ^ (q V r)	p ^ q	p ^ r	(p ^ q) V (p ^ r)	Statement
0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	0	1	0	0	0	0	1
0	1	1	1	0	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	1	1	1	0	1	1	1
1	1	0	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1

Contradiction

A statement that is always false is called a contradiction.

Example: This course is easy `and' this course is not easy

$p \wedge (\neg p) \equiv F$

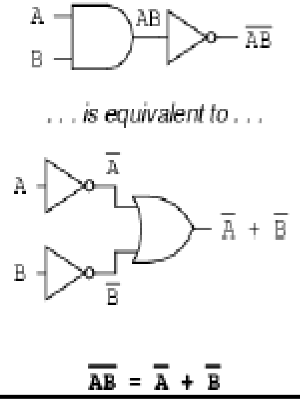
De Morgan's Law

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

p	q	¬p	¬q	p ∧ q	¬(p ∧ q)	¬p ∨ ¬q
T	T	F	F	T	F	F
T	F	F	T	F	T	T
F	T	T	F	F	T	T
F	F	T	T	F	T	T

Augustus De Morgan (1806-1871)



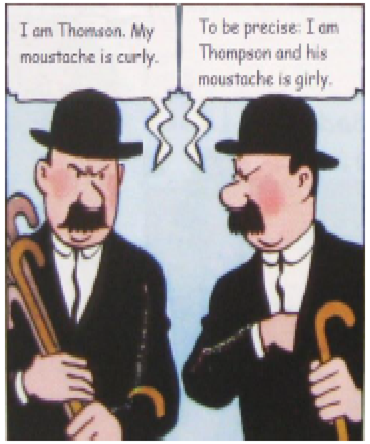
Logical Equivalences

- Useful laws to transform one logical expression to an equivalent one.
- Axioms (T ≡ tautology, C ≡ contradiction):

$\neg T \equiv F$ $\neg F \equiv T$ $\neg T \equiv C \equiv F$ $\neg C \equiv T \equiv T$

- De Morgan: $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- Commutativity:

$p \wedge q \equiv q \wedge p$ $p \vee q \equiv q \vee p$



Logical Equivalence Laws

- double negation: $\neg(\neg p) \equiv p$
- idempotent: $p \wedge p \equiv p$ and $p \vee p \equiv p$
- absorption: $p \vee (p \wedge q) \equiv p$ and $p \wedge (p \vee q) \equiv p$

Quantifiers

- **Universal quantification**
  - $(\forall x)P(x)$ means P holds for **all** values of x in domain associated with variable
  - E.g., $(\forall x)\ dolphin(x) \rightarrow mammal(x)$
- **Existential quantification**
  - $(\exists\ x)P(x)$ means P holds for **some** value of x in domain associated with variable
  - E.g., $(\exists\ x)\ mammal(x) \wedge lays\_eggs(x)$
  - Permits one to make a statement about some object without naming it

Translating English to FOL

Every gardener likes the sun

$\forall x\ gardener(x) \rightarrow likes(x, Sun)$

You can fool some of the people all of the time

$\exists x\ \forall t\ \ person(x) \wedge time(t) \rightarrow can\text{-}fool(x, t)$

You can fool all of the people some of the time

$\forall x\ \exists t\ (person(x) \wedge time(t) \rightarrow\ can\text{-}fool(x, t))$

$\exists t\ \ \forall x\ (person(x) \wedge time(t) \rightarrow\ can\text{-}fool(x, t))$

All purple mushrooms are poisonous

$\forall x\ (mushroom(x) \wedge purple(x)) \rightarrow poisonous(x)$

Modus Ponens & Modus Tollens

Modus Ponens

1. If A, then B.

2. A

So, 3. B.

Example:

1. If it is raining, then the ground is wet.

2. It is raining.

So, 3. The ground is wet

Modus Tollens

1. If A, then B.

2. Not B.

So, 3. Not A.

Example:

1. If everything it says in the Bible is true, then the world was created in six days.

2. The world was not created in six days.

So, 3. Therefore, not everything it says in the Bible is true.

Predicates

- Terms represent specific objects in the world and can be constants, variables or functions.

  Predicate Symbols refer to a particular relation among objects.
- Sentences represent facts, and are made of terms, quantifiers and predicate symbols.

  Functions allow us to refer to objects indirectly (via some relationship).
- Quantifiers and variables allow us to refer to a collection of objects without explicitly naming each object

Example

- Predicates: Brother, Sister, Mother , Father
- Objects: Bill, Hillary, Chelsea, Roger
- Facts expressed as atomic sentences a.k.a.
- literals:

Father(Bill,Chelsea)

Mother(Hillary,Chelsea)

Brother(Bill,Roger)

Unification

We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$.

$\theta = \{x/John, y/John\}$ works

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, SteveJobs)$	$\{x/SteveJobs, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, SteveJobs)$	$fail$

Standardizing variables apart

- *Standardizing apart* eliminates overlap of variables.
- Rename all variables so that variables bound by different quantifiers have unique names.
- For example

  $$\forall x\ Apple(x) \implies Fruit(x)$$
  $$\forall x\ Spider(x) \implies Arachnid(x)$$

  is the same as

  $$\forall x\ Apple(x) \implies Fruit(x)$$
  $$\forall y\ Spider(y) \implies Arachnid(y)$$

Resolution

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

For example,

$$\frac{\begin{array}{l} \neg Rich(x) \vee Unhappy(x) \\ Rich(Ken) \end{array}}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$.

Conjunctive Normal Form (CNF)

- **Resolution works best when the formula is of the special form**: it is an ∧ of ∨s of (possibly negated, ¬) variables (called **literals**).

- This form is called a **Conjunctive Normal Form**, or **CNF**.
  - $(y \vee \neg z) \wedge (\neg y) \wedge (y \vee z)$ is a CNF
  - $(x \vee \neg y \wedge z)$ is not a CNF

- **An AND** $(\wedge)$ **of CNF formulas is a CNF formula.**
  - So if all premises are CNF and the negation of the conclusion is a CNF, then AND of premises AND NOT conclusion is a CNF.

- To convert a formula into a CNF.
  - Open up the implications to get ORs.
  - Get rid of double negations.
  - Convert $F \vee (G \wedge H)$ to $(F \vee G) \wedge (F \vee H)$
    //distributivity

- Example: $A \rightarrow (B \wedge C)$

  $\equiv \neg A \vee (B \wedge C)$

  $\equiv (\neg A \vee B) \wedge (\neg A \vee C)$

- In general, CNF can become quite big, especially when have $\leftrightarrow$. There are tricks to avoid that ...

CNF and DNF

- Every truth table (Boolean function) can be written as either a conjunctive normal form (CNF) or disjunctive normal form (DNF)
- **CNF is an $\wedge$ of $\vee$s**, where $\vee$ is over variables or their negations (literals); an $\vee$ of literals is also called a **clause**.
- **DNF is an $\vee$ of $\wedge$s**; an $\wedge$ of literals is called a **term**.

Why CNF and DNF?

- Convenient normal forms
- **Resolution** works best for formulas in CNF
- Useful for constructing formulas given a **truth table**
  - **DNF**: take a disjunction (that is, $\lor$) of all satisfying truth assignments
  - **CNF:** take a conjunction ($\land$) of **negations** of falsifying truth assignments

- ❑ Any propositional formula is tautologically equivalent to some formula in disjunctive normal form.
- ❑ Any propositional formula is tautologically equivalent to some formula in conjunctive normal form.

Conversion to CNF

1. Eliminate biconditionals and implications.
2. Reduce the scope of $\neg$: move $\neg$ inwards.
3. Standardize variables apart: each quantifier should use a different variable name.
4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a *Skolem function* of the enclosing universally quantified variables.
5. Drop all universal quantifiers: It's allright to do so now.
6. Distribute $\wedge$ over $\vee$.
7. Make each conjuct a separate clause.
8. Standardize the variables apart again.

- All people who are graduating are happy.
  All happy people smile.
  JohnDoe is graduating.

  Is JohnDoe smiling?
- First convert to predicate logic
  $\forall x\, graduating(x) \implies happy(x)$
  $\forall x\, happy(x) \implies smiling(x)$
  $graduating(JohnDoe)$

  $smiling(JohnDoe)$ negate this: $\neg smiling(JohnDoe)$
- Then convert to canonical form.

1. $\forall x\, graduating(x) \implies happy(x)$
2. $\forall x\, happy(x) \implies smiling(x)$
3. $graduating(JohnDoe)$

4. $\neg smiling(JohnDoe)$

Step 1. Eliminate $\implies$

1. $\forall x\, \neg graduating(x) \vee happy(x)$
2. $\forall x\, \neg happy(x) \vee smiling(x)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

1. $\forall x\, \neg graduating(x) \vee happy(x)$
2. $\forall x\, \neg happy(x) \vee smiling(x)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

Step 2. Move $\neg$ inwards. (not needed)

Step 3. Standardize variables apart.

1. $\forall x\, \neg graduating(x) \vee happy(x)$
2. $\forall y\, \neg happy(y) \vee smiling(y)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

1. $\forall x\, \neg graduating(x) \vee happy(x)$
2. $\forall y\, \neg happy(y) \vee smiling(y)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

Step 4. Skolemize. (not needed)

Step 5. Drop all $\forall$.

1. $\neg graduating(x) \vee happy(x)$
2. $\neg happy(y) \vee smiling(y)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

1. $\neg graduating(x) \vee happy(x)$
2. $\neg happy(y) \vee smiling(y)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

Step 6. Distribute $\wedge$ over $\vee$. (not needed)

Step 7. Make each conjuct a separate clause. (not needed)

Step 8. Standardize the variables apart again. (not needed)

Ready for resolution!

1. $\neg graduating(x) \vee happy(x)$
2. $\neg happy(y) \vee smiling(y)$
3. $graduating(JohnDoe)$
4. $\neg smiling(JohnDoe)$

Resolve 4 and 2 using $\theta = \{y/JohnDoe\}$:

5. $\neg happy(JohnDoe)$

Resolve 5 and 1 using $\theta = \{x/JohnDoe\}$:

6. $\neg graduating(JohnDoe)$

Resolve 6 and 3:

7. $\perp$

Resolution

- Resolution is a sound and complete inference procedure for FOL
- Reminder: Resolution rule for propositional logic:
  - $P_1 \vee P_2 \vee \dots \vee P_n$
  - $\neg P_1 \vee Q_2 \vee \dots \vee Q_m$
  - Resolvent: $P_2 \vee \dots \vee P_n \vee Q_2 \vee \dots \vee Q_m$
- Examples
  - P and $\neg P \vee Q$: derive Q (Modus Ponens)
  - $(\neg P \vee Q)$ and $(\neg Q \vee R)$: derive $\neg P \vee R$
  - P and $\neg P$: derive False [contradiction!]
  - $(P \vee Q)$ and $(\neg P \vee \neg Q)$: derive True

Resolution in first-order logic

- Given sentences

  $P_1 \vee \ldots \vee P_n$

  $Q_1 \vee \ldots \vee Q_m$
- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ unify with substitution list θ, then derive the resolvent sentence:

  subst(θ, $P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \ldots P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m$)
- Example
  - from clause $P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$
  - and clause $\neg P(z, f(a)) \vee \neg Q(z)$
  - derive resolvent $P(z, f(y)) \vee Q(y) \vee \neg Q(z)$
  - using $\theta = \{x/z\}$

Example:

Assume:	$E_1 \vee E_2$	playing tennis or raining
and	$\neg E_2 \vee E_3$	not raining or working
Then:	$E_1 \vee E_3$	playing tennis or working

"Resolvent"

General Rule:

Assume:	$E \vee E_{12} \vee \ldots \vee E_{1k}$
and	$\neg E \vee E_{22} \vee \ldots \vee E_{2l}$
Then:	$E_{12} \vee \ldots \vee E_{1k} \vee E_{22} \vee \ldots \vee E_{2l}$

Note: E_{ij} can be negated.

Resolution refutation

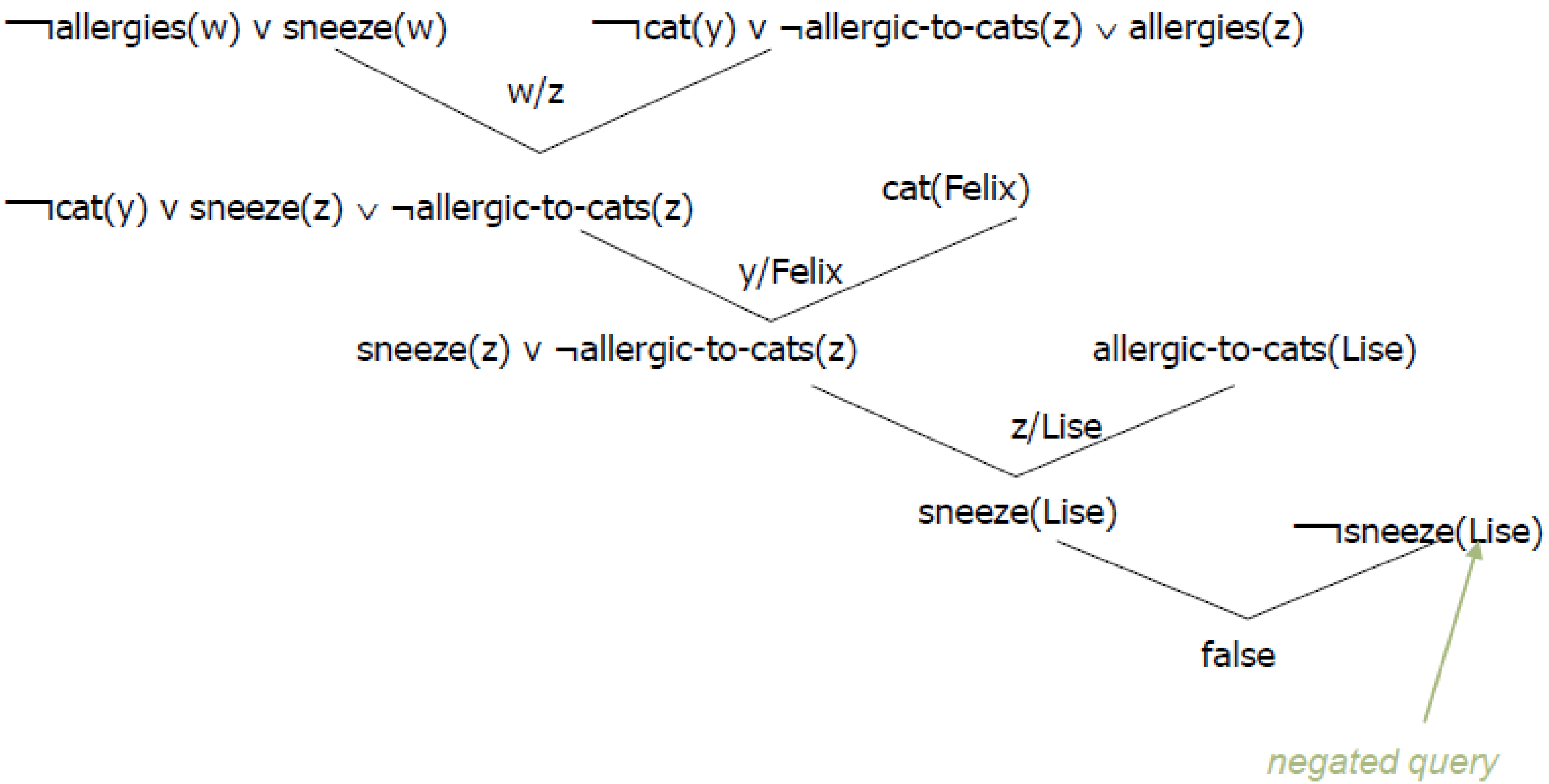
- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- Proof by contradiction: Add $\neg$Q to KB and try to prove false.

  i.e., (KB |- Q) $\leftrightarrow$ (KB $\wedge$ $\neg$Q |- False)
- Resolution is refutation complete: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences

Resolution example

- KB:
  - allergies(X) $\rightarrow$ sneeze(X)
  - cat(Y) $\wedge$ allergic-to-cats(X) $\rightarrow$ allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)
- Goal:
  - sneeze(Lise)

Refutation resolution proof tree



Skolemization

FOL: Conversion to CNF

- Everyone who loves all animals is loved by someone

$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

- Steps to convert to CNF
  - Eliminate biconditionals $\Leftrightarrow$ and implications $\Rightarrow$
  - Move $\neg$ inwards $\neg\forall x\ , p \equiv \exists x \neg p, \neg\exists x\ p \equiv \forall x\ \neg p$
  - Standardize the variables: each quantifier should use a different variable
  - Skolemize: A more general form of existential instantiation
    - Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables
  - Drop universal quantifiers
  - Distribute $\wedge$ over $\vee$

FOL Resolution Example

- Everyone who loves all animals is loved by someone $\forall x[\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
- Anyone who kills an animal is loved by no one $\forall x[\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$
- Jack loves all animals $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$
- Either Jack or Curiosity killed the cat, who is named Tuna $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
- Did Curiosity kill the cat?

$Kills(Curiosity,Tuna)?$

$\forall x\ Cat(x) \Rightarrow Animal(x)$

$Cat(Tuna)$

$$\forall x[\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$$

$$\forall x\,[\neg\forall y\ \neg Animal(y) \lor Loves(x,y)] \lor [\exists y\ Loves(y,x)]$$

$$\forall x\big[\exists y\ \neg\big(\neg Animal(y) \lor Loves(x,y)\big)\big] \lor [\exists y\ Loves(y,x)]$$

$$\forall x\big[\exists y\ \big(Animal(y) \land \neg Loves(x,y)\big)\big] \lor [\exists y\ Loves(y,x)]$$

$$\forall x\big[\exists y\ \big(Animal(y) \land \neg Loves(x,y)\big)\big] \lor [\exists z\ Loves(z,x)]$$

$$\forall x\big[Animal\big(F(x)\big) \land \neg Loves\big(x,F(x)\big)\big] \lor Loves(G(x),x)$$

$$\big[Animal\big(F(x)\big) \land \neg Loves\big(x,F(x)\big)\big] \lor Loves(G(x),x)$$

$$\big[Animal\big(F(x)\big) \lor Loves(G(x),x)\big] \land \big[\neg Loves\big(x,F(x)\big) \lor Loves(G(x),x)\big]$$

$$\forall x[\exists z\ Animal(z) \land Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$$

$$\forall x[\neg\exists z\ Animal(z) \land Kills(x,z)] \lor [\forall y\ \neg Loves(y,x)]$$

$$\forall x\big[\forall z\ \neg\big(Animal(z) \land Kills(x,z)\big)\big] \lor [\forall y\ \neg Loves(y,x)]$$

$$\forall x[\forall z\ \neg Animal(z) \lor \neg Kills(x,z)] \lor [\forall y\ \neg Loves(y,x)]$$

$$\neg Animal(z) \lor \neg Kills(x,z) \lor \neg Loves(y,x)$$

$$\forall x\ Animal(x) \Rightarrow Loves(Jack, x)$$

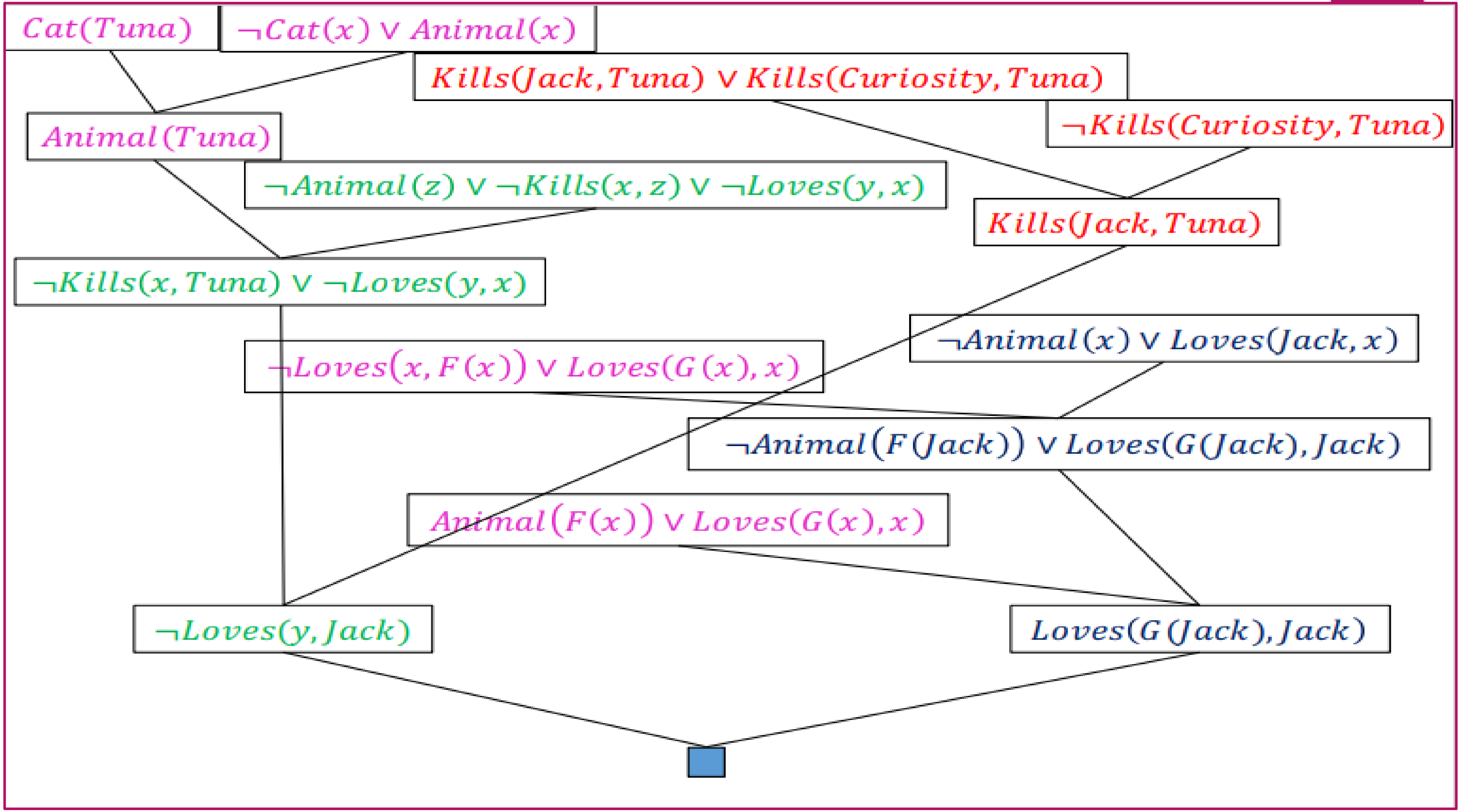
$$\forall x\ \neg Animal(x) \lor Loves(Jack, x)$$

$$\neg Animal(x) \lor Loves(Jack, x)$$

$$\forall x\ Cat(x) \Rightarrow Animal(x)$$

$$\forall x\ \neg Cat(x) \lor Animal(x)$$

$$\neg Cat(x) \lor Animal(x)$$


$Cat(Tuna)$
$\neg Cat(x) \vee Animal(x)$
$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
$\neg Kills(Curiosity, Tuna)$
$Animal(Tuna)$
$\neg Animal(z) \vee \neg Kills(x, z) \vee \neg Loves(y, x)$
$Kills(Jack, Tuna)$
$\neg Kills(x, Tuna) \vee \neg Loves(y, x)$
$\neg Animal(x) \vee Loves(Jack, x)$
$\neg Loves(x, F(x)) \vee Loves(G(x), x)$
$\neg Animal(F(Jack)) \vee Loves(G(Jack), Jack)$
$Animal(F(x)) \vee Loves(G(x), x)$
$\neg Loves(y, Jack)$
$Loves(G(Jack), Jack)$

Example

Jack owns a dog.

Every dog owner is an animal lover.

No animal lover kills an animal.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$
2. $\forall x; \ (\exists y \ \ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
3. $\forall x; \ \ AnimalLover(x) \rightarrow (\forall y \ \ Animal(y) \rightarrow \neg Kills(x, y))$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x : \ \ Cat(x) \rightarrow Animal(x)$

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$

2. $\forall x; \ (\exists y \ \ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$

3. $\forall x; \ \ AnimalLover(x) \rightarrow (\forall y \ \ Animal(y) \rightarrow \neg Kills(x, y))$

4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

5. $Cat(Tuna)$

6. $\forall x : \ \ Cat(x) \rightarrow Animal(x)$

Conjunctive Normal Form

$Dog(D)$

(D is a placeholder for the dogs unknown name (i.e. Skolem symbol/function). Think of D like "JohnDoe")

$Owns(Jack, D)$

$\neg Dog(y) \vee \neg Owns(x, y) \vee AnimalLover(x)$

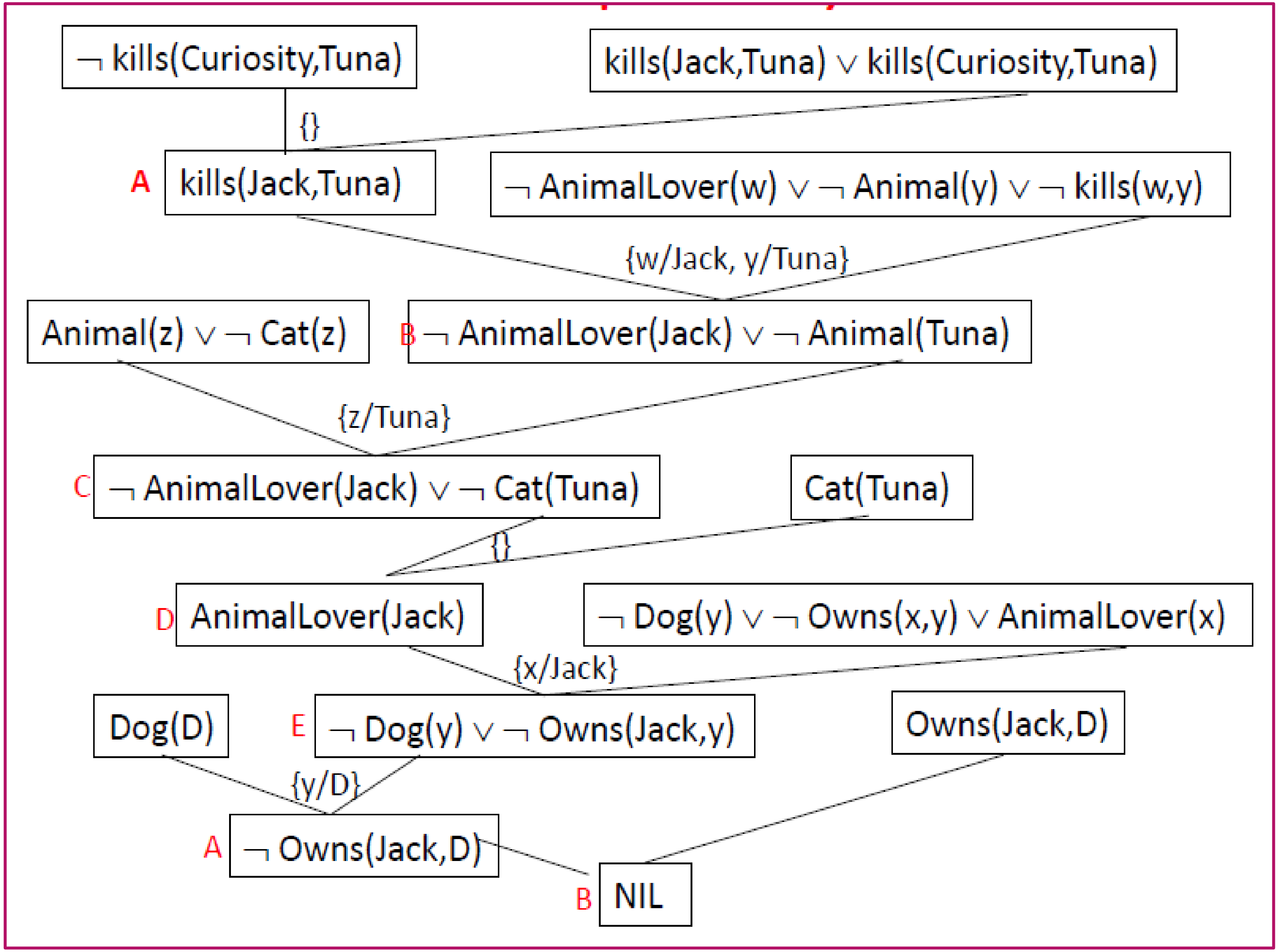
$\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$

$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

$Cat(Tuna)$

$\neg Cat(z) \vee Animal(z)$

$\neg Kills(Curiosity, Tuna)$



Conjunctive Normal Form

$Dog(D)$ (D is a placeholder for the dogs unknown name (i.e. Skolem symbol/function). Think of D like "JohnDoe")

$Owns(Jack, D)$

$\neg Dog(y) \vee \neg Owns(x, y) \vee AnimalLover(x)$

$\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$

$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

$Cat(Tuna)$

$\neg Cat(z) \vee Animal(z)$

$\neg Kills(Curiosity, Tuna)$

Assignment

Knowledge Base in a Natural Language

- 1. Marcus was a man.
- 2. Marcus was a Pompeian.
- 3. All Pompeians were Roman.
- 4. Caesar was a ruler.
- 5. All Romans were either loyal to Caesar or hate Caesar.
- 6. Everyone is loyal to someone.
- 7. People only try to assassinate rulers to whom they are not loyal.
- 8. Marcus tried to assassinate Caesar.
- Query: Does Marcus hate Caesar?

1. Marcus was a man.

- man(Marcus)

2. Marcus was a Pompeian

- pompeian(Marcus)

3. All Pompeians were Romans.

- Vx: pompeian(x) --> roman(x)
- ~pompeian(x) v roman(x)

4. Caesar was a ruler.

- ruler(Caesar)

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

- Vx: roman(x) --> loyalto(x, Caesar) v hate(x, Caesar)
- ~roman(y) v loyalto(y, Caesar) v hate(y, Caesar)

6. Everyone is loyal to someone.

- Vx Ey: loyalto(x, y)
- loyalto(z, f(z))   // f is a skolem function and returns a person that z is loyal to

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

- Vx Vy: man(x) ^ ruler(y) ^ ~loyalto(x, y) --> trytoassassinate(x, y)
- ~(man(a) ^ ruler(b) ^ ~loyalto(a, b)) v trytoassassinate(a, b)
- = ~man(a) v ~ruler(b) v ~loyalto(a, b) v trytoassassinate(a, b)

8. Marcus tried to assassinate Caesar.

- trytoassassinate(Marcus, Caesar)

Introduce 9. ~hated(Marcus, Caesar).

Resolve 9 with 5 unifying Marcus to y yields

10. ~roman(Marcus) v loyalto(Marcus, Caesar)

Resolve 10 with 3 unifying Marcus to x yields

11. ~pompeian(Marcus) v loyalto(Marcus, Caesar)

Resolve 11 with 2 yields

12. loyalto(Marcus, Caesar)

Resolve 12 with 7 unifying Marcus to a and Caesar to b yields

13. ~man(Marcus) v ~ruler(Caesar) v trytoassassinate(Marcus, Caesar)

Resolve 13 with 1 yields

14. ~ruler(Caesar) v trytoassassinate(Marcus, Caesar)

Resolve 14 with 4 yields

15. trytoassassinate(Marcus, Caesar)

Resolve 15 with 8 yields the null hypothesis.

Therefore, ~hate(Marcus, Caesar) is false, so hate(Marcus, Caesar) is true.

It will be Continued....