



# CS-208: Artificial Intelligence

## Topic-17: Resolution in Predicate Logic

# Resolution in Predicate Logic

In propositional logic, it is easy to determine that two literals can not be true at the same time. But in predicate logic it is not easy as the arguments are also need to be considered

Man(Marcus)            and        $\neg$ Man(Marcus)            is a contradiction

Man(Marcus)            and        $\neg$ Man(Spot)            is not a contradiction

So we need a matching process that compares two literals and discover whether there exist a set of substitution that matches them identical.



## Basic idea of Unification

- ✗ Different constant or predicate can not match
- ✓ Identical constant or predicate can match
- ✓ A variable can match with
  - Another variable
  - Any constant
  - predicate expression (except that it should not contain any instance of the variable being matched).

# Complication in finding the consistent substitutions

Finding the consistent substitution is very complex process:



Wrong way of finding the substitutions:

$P(x, x)$	$y/x$	$P(y, x)$	$z/x$	$P(y, z)$
$P(y, z)$		$P(y, z)$		$P(y, z)$

?  $y/x$  and  $z/x$  can not be the correct substitutions because two different variables  $y$  and  $z$  can not be substituted for a same single variable  $x$ .

Right way of finding the substitution:

Find the first substitution and apply to the rest of the arguments and then find other substitution in the similar way

$P(x, x)$	$y/x$	$P(y, y)$	$z/y$	$P(y, z)$
$P(y, z)$		$P(y, z)$		$P(y, z)$

## Objective of the Unification Procedure

The objective is to find **at least one substitution** that causes two literals to match.

Example:

Hate(x, y)

Hate(Marcus, y)

Two sets of substitutions that can match these two literals:


- ✓ Marcus/ x, z/y
- ✓ Marcus/x, y/z

# Unification Algorithm

Unify( $L_1, L_2$ )

// unifies two literals  $L_1$  and  $L_2$

1. If  $L_1$  or  $L_2$  is a variable or constant, then:
  - a) If  $L_1$  and  $L_2$  are identical, then return NIL.
  - b) Else if  $L_1$  is a variable, then if  $L_1$  occurs in  $L_2$  then return FAIL, else return  $\{(L_2 / L_1)\}$ .
  - c) Else if  $L_2$  is a variable, then if  $L_2$  occurs in  $L_1$  then return FAIL, else return  $\{(L_1 / L_2)\}$ .
  - d) Else return FAIL.
2. If the initial predicate symbols in  $L_1$  and  $L_2$  are not identical, then return FAIL.
3. If  $L_1$  and  $L_2$  have a different number of arguments, then return FAIL
4. Set **SUBST** to NIL.
5. For  $i \leftarrow 1$  to number of arguments in  $L_1$  do
  - a) Call Unify with the  $i^{\text{th}}$  argument of  $L_1$  and the  $i^{\text{th}}$  argument of  $L_2$ , putting result in  $S$ .
  - b) If  $S = \text{FAIL}$  then return FAIL.
  - c) If  $S \neq \text{NIL}$  then:
    - i. Apply  $S$  to the remainder of both  $L_1$  and  $L_2$ .
    - ii. **SUBST** := APPEND( $S$ , **SUBST**).
6. Return **SUBST**.

- 
- At the end of the procedure **SUBST** will contain all substitutions used to unify  $L_1$  and  $L_2$ .
  - If **SUBST** is an empty list then it indicates that match was found without any substitution.
  - If **SUBST** contains a single value Fail then it indicates that unification procedure has failed.

# Resolution Procedure in Predicate Logic

**Step1:** Convert all the propositions (axioms) of  $F$  to clause form.

**Step2:** Negate  $S$  and convert the result to clause form. Add it to the set of clauses obtained in Step1.

**Step3:** 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  $\neg T2$  such that one of the parent clauses contains  $T1$  and the other contains  $\neg T2$  and if  $T1$  and  $T2$  are unifiable, then neither  $T1$  nor  $\neg T2$  should appear in 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.

## An illustrative Example for Proving that Marcus hated Caesar using Resolution

English Sentence (Given set of axioms)	Predicate WFF
1) Marcus was a man	$\text{Man}(\text{Marcus})$
2) Marcus was a Pompeian	$\text{Pompeian}(\text{Marcus})$
3) All Pompeian were Romans	$\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x)$
4) Caesar was a ruler	$\text{Ruler}(\text{Caesar})$
5) All Romans were either <u>loyal to Caesar or hate him</u>	$\forall x: \{ \text{Roman}(x) \rightarrow [(\text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})) \wedge \neg(\text{loyalto}(x, \text{Caesar}) \wedge \text{hate}(x, \text{Caesar}))] \}$
6) Everyone is loyal to someone	$\forall x: \exists y \text{ loyalto}(x, y)$
7) Men only try to assassinate ruler they are not loyal to	$\forall x: \forall y: \text{Man}(x) \wedge \text{Ruler}(y) \wedge \text{trytoassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)$
8) Marcus tried to assassinate Caesar	$\text{trytoassassinate}(\text{Marcus}, \text{Caesar})$



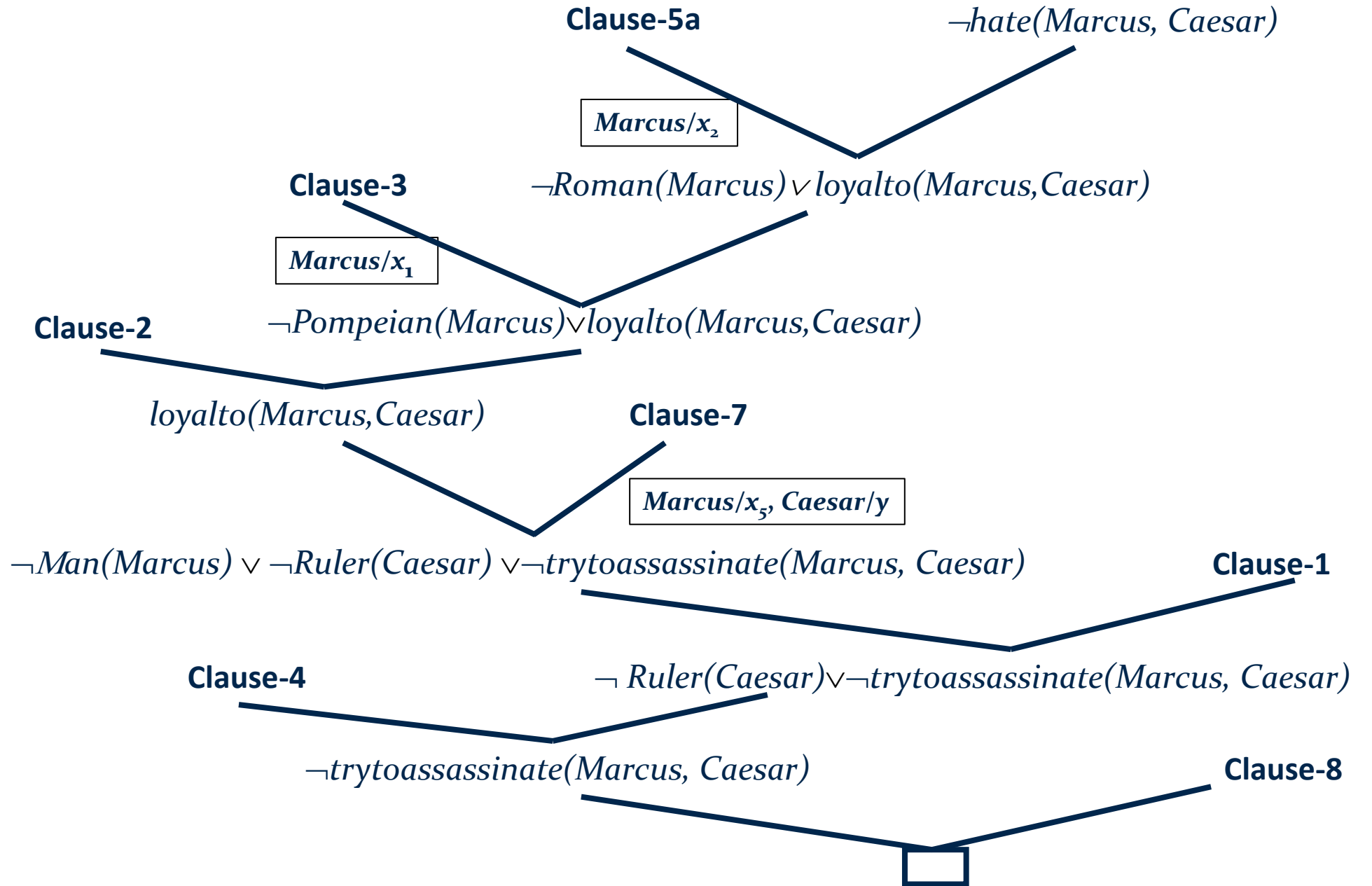
## Resolution Step-1

Predicate WFF	Clauses Form	
Man(Marcus)	Man(Marcus)	Clause-1
Pompeian(Marcus)	Pompeian(Marcus)	Clause-2
$\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x)$	$\neg \text{Pompeian}(x_1) \vee \text{Roman}(x_1)$	Clause-3
Ruler(Caesar)	Ruler(Caesar)	Clause-4
$\forall x: \{ \text{Roman}(x) \rightarrow [(\text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})) \wedge \neg(\text{loyalto}(x, \text{Caesar}) \wedge \text{hate}(x, \text{Caesar}))] \}$	$\neg \text{Roman}(x_2) \vee \text{loyalto}(x_2, \text{Caesar}) \vee \text{hate}(x_2, \text{Caesar})$	Clause-5a
	$\neg \text{Roman}(x_3) \vee \neg \text{loyalto}(x_3, \text{Caesar}) \vee \neg \text{hate}(x_3, \text{Caesar})$	Clause-5b
$\forall x: \exists y \text{ loyalto}(x, y)$	$\text{loyalto}(x_4, \text{S1}(x_4))$	Clause-6
$\forall x: \forall y: \text{Man}(x) \wedge \text{Ruler}(y) \wedge \text{trytoassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)$	$\neg \text{Man}(x_5) \vee \neg \text{Ruler}(y) \vee \neg \text{trytoassassinate}(x_5, y) \vee \neg \text{loyalto}(x_5, y)$	Clause-7
trytoassassinate(Marcus, Caesar)	trytoassassinate(Marcus, Caesar)	Clause-8

## Resolution Proof for Marcus hated Caesar

Statement to be Proved	WFF	Negate the WFF	Clause Form	
<b>Marcus hated Caesar</b>	<code>hate(Marcus, Caesar)</code>	<code>¬hate(Marcus, Caesar)</code>	<code>¬hate(Marcus, Caesar)</code>	Clause-9

# Resolution Proof



# Question Answering using Resolution

It was shown that resolution can be used to answer yes-no questions.

It can also be shown that the resolution can be used to answer fill-in-the-blank questions. This can be done by adding an additional expression to the one which is used to try to find a contradiction. This new additional expression (dummy expression) will simply be the one that is to be proved true (it will be the negation of the expression that is actually used in the resolution).

This dummy expression will not interfere with the resolution process and it is tagged or underlined to indicate that it is a dummy expression. It will carry along the resolution and each time the unification is done, the variables in the dummy expression are also bound just as the ones in the clause that are actively being used.

Instead of terminating on reaching the nil expression, the resolution will terminate when it will only be left with the dummy expression. At this point, the bindings of variables in the dummy expression will provide the answer.

# An illustrative example for answering fill-in-the-blank questions by using resolution

When did Marcus die?

To find answer	WFF	Negate	Adding the dummy Clause	Beginning parent Clause
When did Marcus die?	$\text{died}(\text{Marcus}, t)$	$\neg \text{died}(\text{Marcus}, t)$	$\neg \text{died}(\text{Marcus}, t) \vee \underline{\text{died}(\text{Marcus}, t)}$	$\neg \text{died}(\text{Marcus}, t) \vee \underline{\text{died}(\text{Marcus}, t)}$

## Translating the Sentences into Formulas in Predicate Logic.

English Sentence Given set of axioms	WFF
Marcus was a man	$\text{Man}(\text{Marcus})$
Marcus was a Pompeian	$\text{Pompeian}(\text{Marcus})$
All Pompeian died when the volcano erupted in 79AD	$\text{erupted}(\text{volcano}, 79) \wedge \forall x: \text{Pompeian}(x) \rightarrow \text{died}(x, 79)$

## Convert the Formulas into Clause Form

WFF	Clause Form	
Man(Marcus)	Man(Marcus)	Clause-1
Pompeian(Marcus)	Pompeian(Marcus)	Clause-2
erupted(volcano, 79) $\wedge$ $\forall x$ :Pompeian(x) $\rightarrow$ died(x, 79)	erupted(volcano, 79)	Clause-3
	$\neg$ Pompeian(x) $\vee$ died(x, 79)	Clause-4

# Comparative example for answering yes-no & fill-in-the-blank question by using Resolution

## Answering yes-no question using resolution

**died(Marcus, t)**

$\neg \text{Pompeian}(x) \vee \text{died}(x, 79)$        $\neg \text{died}(\text{Marcus}, t)$

**Marcus/x, 79/t**

$\text{Pompeian}(\text{Marcus})$        $\neg \text{Pompeian}(\text{Marcus})$

Nil

Answer: Marcus died is true

## Answering fill-in-the-blank question using resolution

**died(Marcus, t)**

$\neg \text{Pompeian}(x) \vee \text{died}(x, 79)$        $\neg \text{died}(\text{Marcus}, t) \vee \text{died}(\text{Marcus}, t)$

**Marcus/x, 79/t**

$\text{Pompeian}(\text{Marcus})$        $\neg \text{Pompeian}(\text{Marcus}) \vee \text{died}(\text{Marcus}, 79)$

died(Marcus, 79)

Answer: Marcus died in 79 AD