# Using Predicate Logic

## CSE-345: Artificial Intelligence

# Introduction

- Representing Facts-the language of logic
- Logical formalism is appealing because it immediately suggests a powerful way of deriving new knowledge from old

  -mathematical deduction

- A new statement is true by proving that it follows from the statements that are already known.
- Deductions as a way of deriving answers to questions & solutions to problems

**Logic symbols:**

- $\rightarrow$ (material implication)
- $\neg$ (not)
- $\vee$ (or)
- $\wedge$ (and)
- $\forall$ (for all)
- $\exists$ (there exist)

# Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines the sentences in the language
- Semantics define the "meaning" of sentences;

  i.e., define truth of a sentence in a world

- E.g., the language of arithmetic
  - $x + 2 \geq y$ is a sentence; $x_2 + y >$ is not a sentence
  - $x + 2 \geq y$ is true in the number $x + 2$ is no less than the number $y$
  - $x + 2 \geq y$ is true in a world where $x=7$; $y=1$
  - $x + 2 \geq y$ is false in a world where $x=0$; $y=6$

# Entailment

- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- Knowledge base (*KB*) entails sentence α if and only if α is true in all worlds where *KB* is true
  - E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"
  - E.g., $x+y = 4$ entails $4 = x+y$
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

# Models

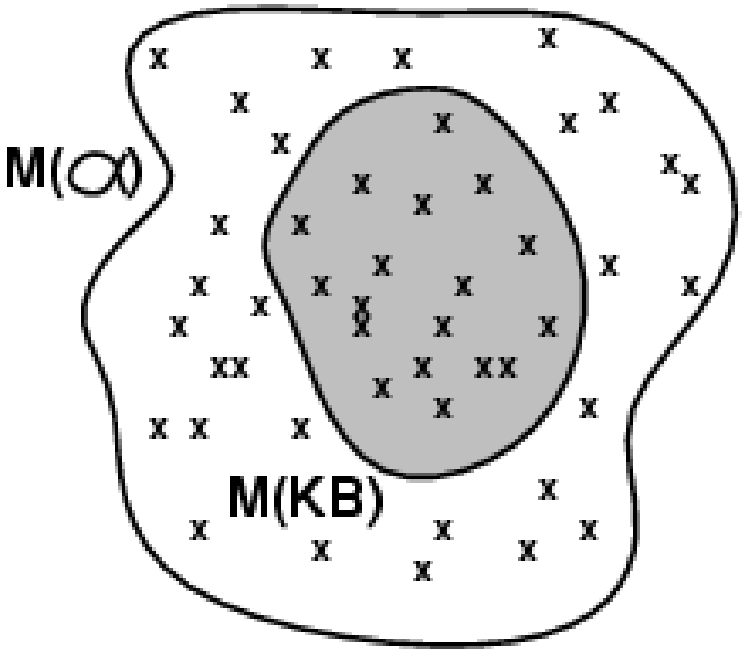
- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

  - We say *m* is a model of a sentence α if α is true in *m*

  *M(α)* is the set of all models of α

  Then KB $\models$ α iff $M(KB) \subseteq M(\alpha)$

  E.g. *KB* = Giants won and Reds won α = Giants won



# Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols $P_1$, $P_2$ etc are sentences

  - If S is a sentence, $\neg S$ is a sentence (negation)
  - If $S_1$ and $S_2$ are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

# Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
false true false

(With these symbols, 8 possible models, can be enumerated automatically)

Rules for evaluating truth with respect to a model *m*:

| | | | |
|---|---|---|---|
| $\neg S$ | is true iff | $S$ is false | |
| $S_1 \wedge S_2$ | is true iff | $S_1$ is true and | $S_2$ is true |
| $S_1 \vee S_2$ | is true iff | $S_1$is true or | $S_2$ is true |
| $S_1 \Rightarrow S_2$ | is true iff | $S_1$ is false or | $S_2$ is true |
| | is false iff | $S_1$ is true and | $S_2$ is false |
| $S_1 \Leftrightarrow S_2$ | is true iff | $S_1 \Rightarrow S_2$ is true and | $S_2 \Rightarrow S_1$ is true |

Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

# Truth tables for connectives

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \Rightarrow Q$ | $P \Leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| *false* | *false* | *true* | *false* | *false* | *true* | *true* |
| *false* | *true* | *true* | *false* | *true* | *true* | *false* |
| *true* | *false* | *false* | *false* | *true* | *false* | *false* |
| *true* | *true* | *false* | *true* | *true* | *true* | *true* |

# Logical equivalence

- Two sentences are logically equivalent iff true in same models: $\alpha \equiv$ ß iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

# Validity and Satisfiability

- A sentence is valid if it is true in all models,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

- Validity is connected to inference via the Deduction Theorem:

  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

- A sentence is satisfiable if it is true in some model

  e.g., $A \vee B$, $C$

- A sentence is unsatisfiable if it is true in no models

  e.g., $A \wedge \neg A$

- Satisfiability is connected to inference via the following:

  $KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

# Representing Simple Facts in Logic

- Propositional Logic (PPL): a way of representing real world knowledge
- Appealing because it is simple to deal with & a decision procedure for it exists
- Real world facts can be represented easily using prepositions logic
- Written as well-formed formulas (wff's)

# An Example

- It is raining: RAINING
- It is sunny: SUNNY
- It is windy: WINDY
- If it is raining, then it is not sunny

$RAINING \rightarrow \neg SUNNY$

# Limitation of PPL

- Socrates is a man: SOCRATESMAN
- Plato is a man: PLATOMAN
- Which would be a totally separate assertion,
  - not be able to draw any conclusions about the similarities between Socrates & Plato
- Much better

MAN(SOCRATES)

MAN(PLATO)

Structure of the representation reflects the structure of the knowledge itself

# Limitation of PPL

- To use predicates applied to arguments.
- More difficult:

All men are mortal: MORTALMAN

- fails to capture the relationship between any individual being a man & that individual being a mortal
- need variables & quantification unless we are willing to write separate statements about the morality of every known man

- Predicate logic (First order predicate logic-FOPL) is needed to represent fact

# Predicate Logic

- Providing a way of deducing new statements from old ones
- Not decidable, semi-decidable
- Despite the theoretical undecidability of PL, it can still serve as a useful way of representing & manipulating some of the kinds of knowledge

# Predicate Logic: Example

1. Marcus was a man
2. Marcus was a Pompeian
3. All Pompeian's were Roman
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

- **Was Marcus loyal to Caesar?**

**Answer Please!!!**

1. Marcus was a man
2. Marcus was a Pompeian
3. All Pompeian's were Roman
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

# Predicate Logic

1. man (Marcus)
2. Pompeian (Marcus)
3. $\forall$x: Pompeian(x)$\rightarrow$ Roman (x)
4. ruler (Caesar)
5. $\forall$x : Roman (x)$\rightarrow$ loyal to (x, Caesar) $\vee$ hate (x, Caesar) **[Inclusive or-OR]**

   $\forall$x: Roman (x)$\rightarrow$[(loyal to (x, Caesar) $\vee$ hate (x, Caesar)) $\wedge$ $\neg$(loyal to (x, Caesar) $\wedge$ hate (x, Caesar))] **[Exclusive or-XOR]**
6. $\forall$x: $\exists y$ : loyal to (x,y)
7. $\forall$x: $\exists y$ : Person (x) $\wedge$ ruler (y) $\wedge$ tryassassinate (x,y)$\rightarrow$ $\neg$ loyalto (x,y)
8. tryassassinate (Marcus, Caesar)

# Predicate Logic: Example

Q: Was Marcus loyal to Caesar?

¬loyalto(Marcus, Caesar)

- In order to prove the goal, we need to use the rules of inference to transform it into another goal (or set of goals) that can in turn be transformed,

- so on, until there are no unsatisfied goals remaining.

- AND-OR graph may be used

  -when there are alternative ways of satisfying individual goals

# Predicate Logic: Example

- Attempt to produce a proof of the goal by reducing the set of necessary but as yet unattained goals to the empty set.
- Fails to satisfy the goal *person (Marcus)*

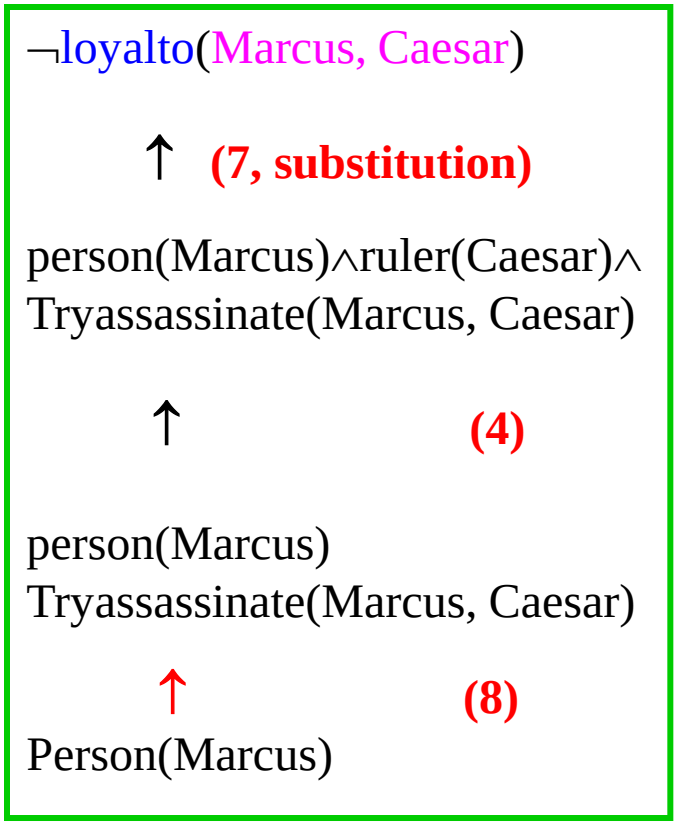



Fig. an attempt to prove ¬loyalto(Marcus, Caesar)

# Predicate Logic: Example

- *person (Marcus):* no info is given
- Marcus was a man & Marcus was a person
- Add the representation of another fact to our system:

All men are people

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

- Now we can satisfy the last goal & produce a proof that Marcus was not loyal to Caesar

# 03 Important Issues

1. Many English sentences are ambiguous (5/6/7). Choosing the correct interpretation may be difficult
2. There is often a choice of how to represent the knowledge. Simple representation is desirable, but they may preclude certain kinds of reasoning. The expedient representation for a particular set of sentences depends on the use to which the knowledge contained in the sentences will be put
3. Even in very simple situations, a set of sentences is unlikely to contain all the info necessary to reason about the topic at hand. In order to able to use a set of statements effectively, it is usually necessary to have access to another set of statements that represents facts that people consider too obvious to mention.

# Computable Functions & Predicates

- All simple facts expressed as combinations of individual predicates

  tryassassinate (Marcus, Caesar)

- Fine if the number of facts is not very large or if the facts themselves are sufficiently unstructured that there is little alternative

# Computable Functions & Predicates

- Greater-than & less-than relationship

  gt(1,0)  lt(0,1)

  gt(2,1)  lt(1,2)

  gt(3,2)  lt(2,3)

  .  .

  -clearly we do not want to have to write out the representation of each of these facts individually

  -there are infinitely many of them

  -but even if we only consider the finite number of them that can be represented,

  -using a single machine word/number, it would be extremely inefficient to store explicitly a large set of statements

  -Thus it becomes useful to augment representation by these *computable predicates*

# Computable Functions & Predicates

- We can simply invoke a procedure, which we will specify in addition to our regular rules, that will evaluate it & return true/false
- It is often useful to have computable functions as well as computable predicates.

*gt*(2+3,1)

- First compute the value of the plus function & then send the arguments 5 & 1 to *gt*

# Computable Functions & Predicates:

## Example

1. Marcus was a man: $man(Marcus)$
2. Marcus was a Pompeian: $Pompeian(Marcus)$
3. Marcus was born in 40 A. D.: $born(Marcus, 40)$
4. All men are mortal: $\forall x: man(x) \rightarrow mortal(x)$
5. All Pompeians died when the volcano erupted in 79 A. D.

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

6. No mortal lives longer than 150 years

$\forall x : \forall t_1 : \forall t_2 : mortal\ (x) \wedge born(x,t1) \wedge gt(t_2\text{-}t_1,150) \rightarrow dead(x,t_2)$

7. It is now 2016: $now = 2016$
8. Alive means not dead

$\forall x : \forall t_1 : [alive(x,t) \rightarrow \neg\, dead(x,t)] \wedge [\neg\, dead(x,t) \rightarrow alive\ (x,t)$

9. If someone dies, then he is dead at all later times

$\forall x : \forall t_1 : \forall t_2 : died(x,t_1) \wedge gt(t_2, t_1) \rightarrow dead(x,t_2)$

# Computable Functions & Predicates

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $x: man(x) \rightarrow mortal(x)$
5. $\forall x: [Pompeian(x) \rightarrow died(x,79)]$
6. $erupted(volcano, 79)$
7. $\forall x : \forall t_1 : \forall t_2 : mortal\ (x) \wedge born(x,t1) \wedge gt(t_2\text{-}t_1,150) \rightarrow dead(x,t_2)$
8. $now = 2016$
9. $\forall x : \forall t_1 : [alive(x,t) \rightarrow \neg\, dead(x,t)] \wedge [\neg\, dead(x,t) \rightarrow alive\ (x,t)]$
10. $\forall x : \forall t_1 : \forall t_2 : died(x,t_1) \wedge gt(t_2, t_1) \rightarrow dead(x,t_2)$

# Computable Functions & Predicates

- Is Marcus alive?

¬alive(Marcus, now)

*¬alive(Marcus, now)*

↑ (9, substitution)

*Dead(Marcus, now)*

↑ (10, substitution)

*Died(Marcus, t1)∧ gt(now,t1)*

↑ (5, substitution)

*Pompeian(Marcus) ∧ gt(now,79)*

– (2)

*gt(now,79)*

↑ *(8, substitute equals)*

*gt(2016, 79)*

↑ (compute gt)

*nil*

**nil**: list of conditions remaining to be proved is empty
-the proof has succeeded

**Two things:**

1. Even every simple conclusions can require many steps to prove
2. A variety of processes (matching, Substitution, modus pones) are involved in the production of proof.

# NEXT CLASS

# Resolution

- A procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.
- Resolution produces proofs by refutation
- To prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statement

# Conversion to Clause Form

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

$\forall x$: [Roman(x) $\wedge$know(x,Marcus)]

$\rightarrow$[hate(x,Caesar) $\wedge(\forall y: \exists z : \text{hate}(y,z) \rightarrow \text{thinkcrazy}(x,y))]$

-to use this formula in a proof requires a complex matching process

-if the formula were in a simpler form, this process would be much easier.

-the formula would be easier to work with if

- it were flatter, i.e., there was less embedding of components
- the quantifiers were separated from the rest of the formula so that they did not need to be considered

# Conversion to Clause Form

- Conjunctive normal form (CNF) has both of these properties.

**CNF**: ¬Roman(x)∨ ¬know(x,Marcus) ∨

hate(x,Caesar) ∨ ¬hate(y,z) ∨ thinkcrazy(x,z)

- For Resolution, need to reduce a set of wff's to a set of clauses

  -a clause is defined to be a wff in CNF but with no instances of the connector ∨

  -to do this, by first converting each wff into conjunctive normal form

  -then breaking apart each such expression into clauses, one for each conjunct.

# Algorithm: Convert to Clause Form

1. Eliminate $\rightarrow$, using the fact that $a \rightarrow b$ is equivalent to $\neg a \vee b$

$\forall x: \neg[\text{Roman}(x) \wedge \text{know}(x,\text{Marcus})] \vee$

$\text{hate}(x,\text{Caesar}) \vee (\forall y: \neg(\exists z: \text{hate}(y,z)) \vee \text{thinkcrazy}(x,y))]$

2. Reduce the scope of each $\neg$ to a single term, using the fact that $\neg(\neg p)=p$, deMorgan's Law as

$[\neg(a \wedge b)= \neg a \vee \neg b$ and $\neg(a \vee b)= \neg a \wedge \neg b]$

Correspondences between quantifiers

$[\neg \forall x: P(x)= \exists x: \neg P(x)$ and $\neg \exists x: P(x)= \forall x: \neg P(x)]$

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

$\text{hate}(x,\text{Caesar}) \vee (\forall y: \forall z: \neg(\text{hate}(y,z)) \vee \text{thinkcrazy}(x,y))]$

# Algorithm: Convert to Clause Form

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

$\forall x: P(x) \vee \forall x: Q(x)$ converted to

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

4. Move all quantifiers to the left of the formula without changing their relatives order.

$\forall x: \forall y: \forall z: [\neg Roman(x) \vee \neg know(x,Marcus)] \vee$

$[hate(x,Caesar) \vee (\neg\ hate(y,z) \vee thinkcrazy(x,y))]$

# Algorithm: Convert to Clause Form

5. Eliminate existential quantifiers.

   $\exists x$ : *President(x)* transformed to *President(S1)*

S1-function with no arguments that somehow produces a value that satisfies President

6. Drop the prefix

[¬Roman(x) ∨ ¬know(x,Marcus)] ∨

[hate(x,Caesar) ∨ (¬ hate(y,z) ∨ thinkcrazy(x,y))]

# Algorithm: Convert to Clause Form

7. Convert the matrix into a conjunction of disjuncts . Associative: [a ∨ (b ∨ c)=(a ∨ b) ∨ c] & remove the parentheses

¬Roman(x) ∨ ¬know(x,Marcus) ∨

hate(x,Caesar) ∨ ¬ hate(y,z) ∨ thinkcrazy(x,y)

- Distributive: [(a ∧ b) ∨ c = (a ∨ c) ∧(b∨ c)]

(winter∧wearingboots) ∨ (summer ∧ wearingsandals)

-[winter ∨ (summer ∧ wearingsandals)] ∧ [wearingboots ∨ (summer ∧ wearingsandals)]

-(winter ∨ summer) ∧

(winter ∨ wearingsandals) ∧

(wearingboots ∨ summer) ∧

(wearingboots ∨ wearingsandals)

# Algorithm: Convert to Clause Form

8. Create a separate clause corresponding to each conjunct. In order for wff's to be true, all the clauses that are generated from it must be true.
9. Standardize apart the variables in the set of clauses generated in step 8.

-rename the variables so that no two clauses make reference to the same variable.

$(\forall x: P(x) \wedge Q(x)) = \forall x: P(x) \wedge \forall x: Q(x)$

- After applying entire procedure to a set of wff's, we will have a set of clauses each of which is a *disjunction of literals*.

# The Basis of Resolution

- The resolution procedure is a simple iterative process: at each step, two clauses (parent clause) are compared (resolved), yielding new clause that has been inferred from them.
- The new clause represents ways that the two parent clauses interact with each other.

$$winter \vee summer$$

$$\neg\ winter \vee cold$$

-both clauses must be true (the clauses, although they look independent, are really conjoined)

-if *winter* is true, then *cold* must be true to guarantee the truth of the 2nd clause

-if $\neg$ *winter* is true, the *summer* must be true to guarantee the truth of the 1st clause

We can Deduce: $summer \vee cold$

# The Basis of Resolution

- Resolution operates by taking two clauses that each contain the same literal (*winter*)
- The literal must occur in positive form in one clause & in negative form in the other
- The *resolvent* is obtained by combining all of the literals of the two parent clauses except the ones that cancel
- If the clause that is produced is the empty clause, then a contradiction has been found.

*winter*

*¬winter*

-will produce empty clause

# Basis of Resolution in Predicate Logic

- Herbrand's theorem [Chang and Lee, 1973]

1. To show that a set of clauses *S* is unsaisfiable, it is necessary to consider only interpretation over a particular set, called the *Herbrand universe of S*
2. A set of clauses S is unsaisfiable if and only if a finite subset of ground instances (in which all bound variables have had a value substituted for them) of S is unsaisfiable.

# Algorithm: Propositional Resolution

1. Convert all the propositions of F to clause form
2. Negate P & convert the result to clause form. Add it to the set of clauses obtained in step 1
3. Repeat until either a contradiction is found or no progress can be made:
   a) Select two clauses. Call these the parent clauses
   b) Resolve them together (Resolvent).

   Exception: if there are any pairs of literals L & ¬L such that one of the parent clauses contains L & the other contains ¬L, then select one such pair & eliminate both.

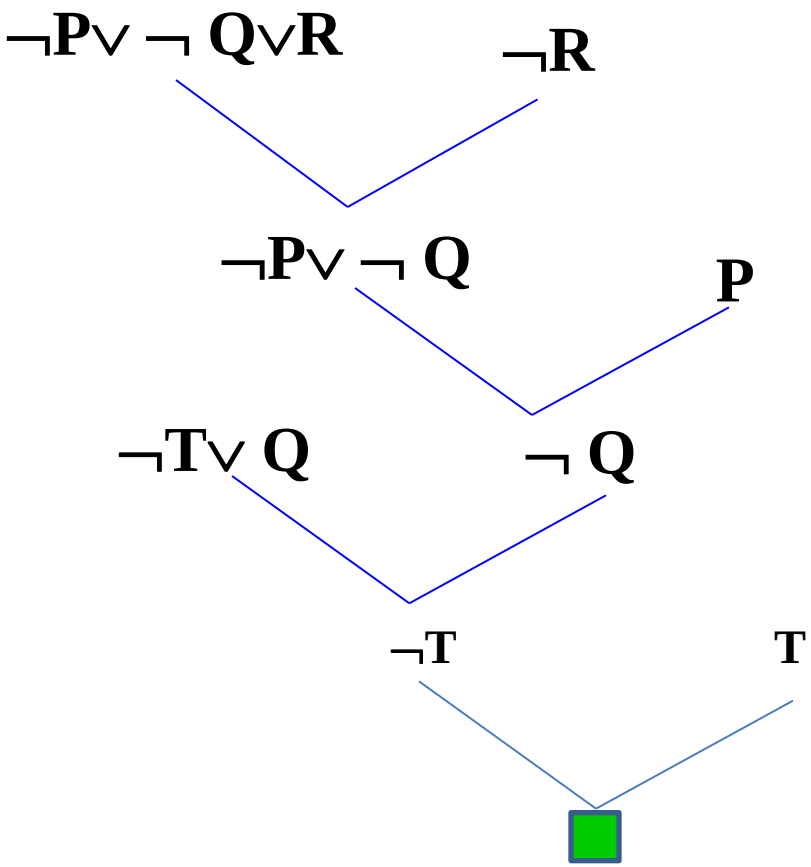
   c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure

# Example: 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** | **(5)** |

-First convert the axioms to clause form, then negate R for ¬R
-then selecting pairs of clauses to resolve together

# Example: prove R




-There is no way for all of these clauses to be true in a single interpretation
-This is indicated by the empty clause

# Unification

- In PPL, it is easy to determine that two literals cannot both be true at the same time.
- In Predicate Logic, matching process is more complicated since the arguments of the predicates must be considered

- man(John) and ¬man(John) is a contradiction

While man(John) and ¬man(Spot) is not

# Unification

- In order to determine contradictions,

  -need a matching procedure that compares two literals

  - discovers whether there exists a set of substitutions that makes them identical

- There is a straightforward recursive procedure, called *unification algorithm*.

- **Idea:** first check if their initial predicate symbols are the same

  -if so, proceed. Otherwise, there is no way they can be unified, regardless of their arguments.

# Unification

*tryassassinate(Marcus, Caesar)*

*hate(Marcus, Caesar)*

-cannot be unified

-if the predicate symbols match, then check the arguments one pair at a time

-if the first matches, continue with the 2nd, & so on

-to test each argument pair, simply call the unification procedure recursively

-different constants/predicates cannot match; identical ones can

-A variable can match another variable, any constant, or a predicate expression, with the restriction that the predicate expression must not contain any instances of the variable being match

# Unification

- Complication: must find a single, consistent substitution for the entire literal, not separate ones for each piece of it.
- To do this, we must take each substitution that we find & apply it to remainder of the literals before we continue trying to unify them.

P(x,x)

P(y,z)

-two instances of P match, fine

-next compare x & y,

-decide that if we substitute y for x, they could match

-y/x

-z/x

-cannot substitute both y & z for x

-no consistent substitution has produce

# Unification

-after first substitution

P(y,y)

P(y,z)

-attempt to unify arguments y & z, which succeeds with substitution z/y

-entire unification process has now succeeded with a substitution that is the composition of the two substitutions.

(z/y)(y/x)

-(a1/a2, a3/a4,….)(b1/b2, b3/b4,……)…means to apply most list, then take the result & apply all ones of the next list, & so forth, until all substitutions have been applied

# Unification

- The object of unification is to discover at least one substitution that causes two literals to match
- Usually, if there is one such substitution there are many

hate(x,y)

hate(Marcus, z)

Could be unified:

(Marcus/x, z/y)

(Marcus/x, y/z)

(Marcus/x, Caesar/y, Caesar/z)

(Marcus/x, Polonius/y, Polonius/z)

-first 02 are equivalent except for lexical variation

-2nd two, although they produce a match, also produce a substitution that is more restrictive than absolutely necessary for the match

# Algorithm: Unify (L1,L2)

1. If L1 or L2 are both variables or constants, then:
   a) If L1 & 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 in L1 then return {FAIL}, else return (L1/L2)
   d) Else return {FAIL}
2. If the initial predicate symbols in L1 & L2 are not identical, then return {FAIL}
3. If L1 & 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 & L2)

# Algorithm: Unify (L1,L2)

5. For i ← 1 to number of arguments in L1:
   a) Call Unify with the $i^{th}$ argument of L1 & 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 & L2
      ii. SUBST: = APPEND(S, SUBST)
6. Return SUBST

# Resolution in Predicate Logic

- Easy way of determining that two literals are contradictory—they are if one of them can be unified with the negation of the other
  - man(x) and ¬man(Spot) are contradictory

    -since man(x) & man(Spot) can be unified

1. man(Marcus)
2. ¬ man($x_1$)∨mortal($x_1$)

-the literal man(Marcus) can be unified with man($x_1$): Marcus/$x_1$ (for $x_1$=Marcus, ¬ man(Marcus) is false

-we cannot cancel out the two man literals.

-clause 2 says that for a given $x_1$, either ¬ man($x_1$) or mortal ($x_1$).

So, for it to be true, we can now conclude only that

mortal (Marcus) must be true.

-the resolvent generated by Clauses 1 & 2 must be mortal(Marcus), which we get by applying the result of the unification process to the resolvent

# Algorithm: Resolution

Assuming a set of given statements F & a statement to be proved P:

1. Convert all the statements of F to clause form
2. Negate P & 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 expected.

a) select two clauses (parent clause)

b) Resolve them together

-the resolvent will be the disjunction of all the literal of both parent clauses with appropriate substitutions performed

# Algorithm: Resolution

**Exception:** if there is one pair of literals T1 & T2 such that one of the parent clauses contains T1 & the other contains T2 &

if T1 & T2 are unifiable, then neither T1 nor T2 should appear in the resolvent.

-We call T1 & T2 *complementary literals*

-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.

# Strategies for Speed Up Process

- If the choice of clauses to resolve together at each step is made in certain systematic ways, then the resolution procedure will find a contradiction if one exists.
- However, it may take a very long time.

1. Only resolve pairs of clauses that contain complementary literals, since only such resolutions produce new clauses that are harder to satisfy than their parents.

To facilitate this, index clauses by the predicates they contain, combined with an indication of whether the predicate is negated.

Then, a given a particular clause, possible resolvents that contain a complementary occurrence of one of its predicates can be located directly.

# Strategies for Speed Up Process

2. Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions. Two kinds of clauses should be eliminated: tautologies (which can never be unsatisfied) and clauses that are subsumed by other clauses (they are easier to satisfy).

   -P V Q is subsumed by P

3. Whenever possible, resolve either with one of the clauses that is part of the statement we are trying to refute or with a clause generated by a resolution with such a clause.

This is called the *set-of-support strategy* & corresponds to the intuition that the contradiction we are looking for must involve the statement we are trying to prove.

Any other contradiction would say that the previously believed statements were inconsistent

# Strategies for Speed Up Process

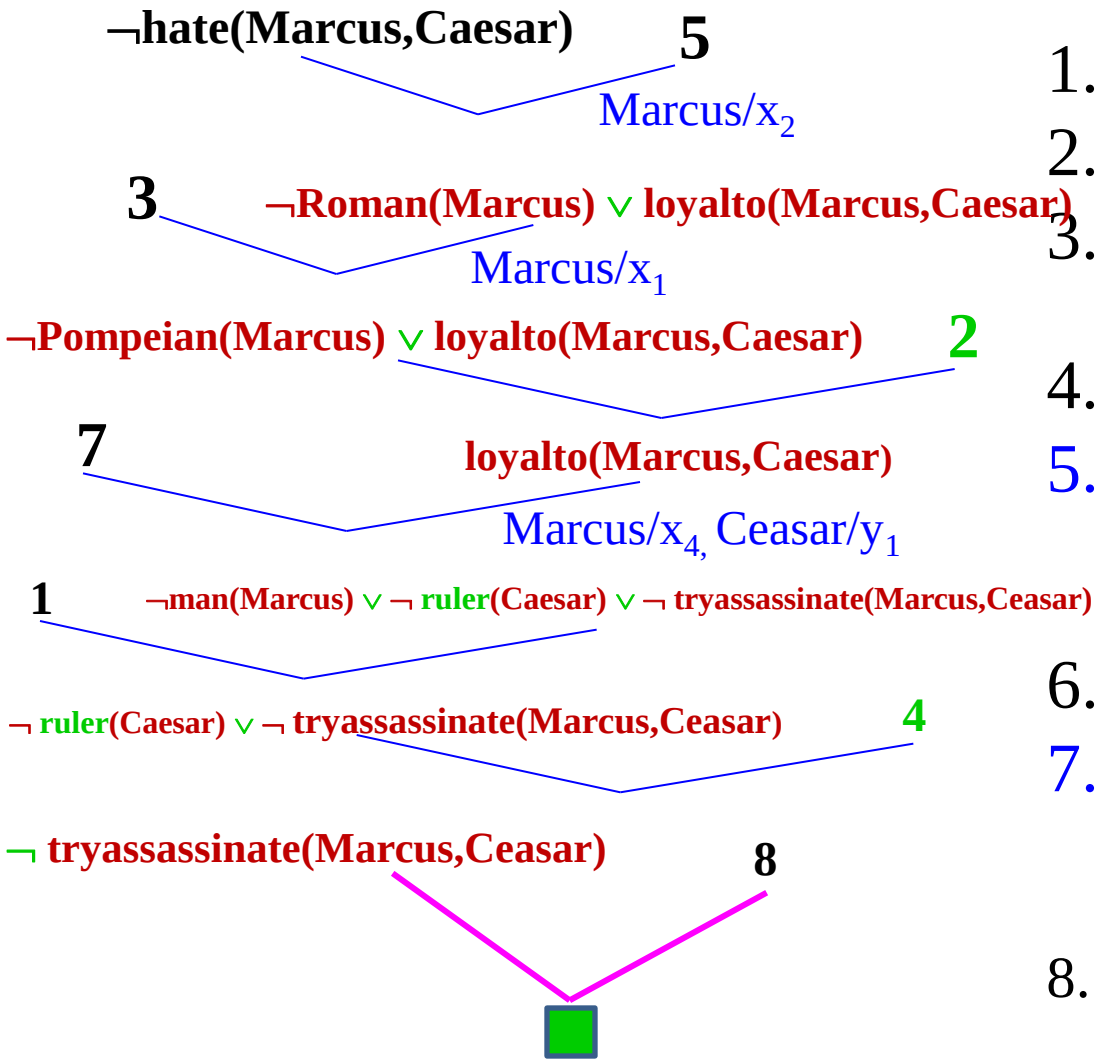
4. Whenever possible, resolve with clauses that have a single literal.

Such resolutions generate new clauses with fewer literals than the larger of their parent clauses & thus are probably closer to the goal of a resolvent with zero items.

-This method is called the *unit-preference strategy*.

# An Example

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




1. man(Marcus)
2. Pompeian(Marcus)
3. $\neg$Pompeian($x_1$) $\vee$Roman($x_1$)
4. ruler(Caesar)
5. $\neg$ Roman($x_2$) $\vee$ loyalto($x_2$, Caesar) $\vee$ hate($x_2$, Caesar)
6. loyalto($x_3$, f1($x_3$))
7. $\neg$man($x_4$) $\vee$ $\neg$ ruler($y_1$) $\vee$ $\neg$ tryassassinate($x_4$,$y_1$) $\vee$ loyalto($x_4$,$y_1$)
8. tryassassinate(Marcus,Caesar)

# An Example

- *hate*(Maucus, Caesar)
- Did Marcus hate Caesar?

*Prove:* $\neg$ *hate*(Maucus, Caesar)

Add: *hate*(Maucus, Caesar) to the set of available clauses & begun resolution process

-no clauses that contain a literal $\neg$ *hate*

*-since the resolution process can only generate new clauses that are composed of combinations of literals form already existing clauses, we know that no such clause can be generated & thus we conclude that*

$\neg$ *hate(Marcus, Caesar) will not produce a contradiction with the known statements.*

# Unsuccessful attempt at Resolution

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

**¬loyalto(Marcus,Caesar)** **5**

Marcus/$x_2$

**3** **¬Roman(Marcus) ∨ hate(Marcus,Caesar)**

Marcus/$x_1$

**¬Pompeian(Marcus) ∨ hate(Marcus,Caesar)** **2**

**hate(Marcus,Caesar)**

Marcus/$x_4$, Ceasar/$y_1$

9. persecute(x,y)→hate(y,x)

10. hate(x,y) → persecute(y,x)

9. ¬persecute($x_5$,$y_2$)∨hate($y_2$,$x_5$)

10. ¬ hate($x_6$,$y_3$) ∨ persecute($y_3$,$x_6$)

**hate(Marcus,Caesar)** **10**

Marcus/$x_6$,Caesar/$y_3$

**¬persecute(Marcus,Caesar)** **9**

Marcus/$x_5$, Caesar/$y_2$

**hate(Marcus,Caesar)**

-

-

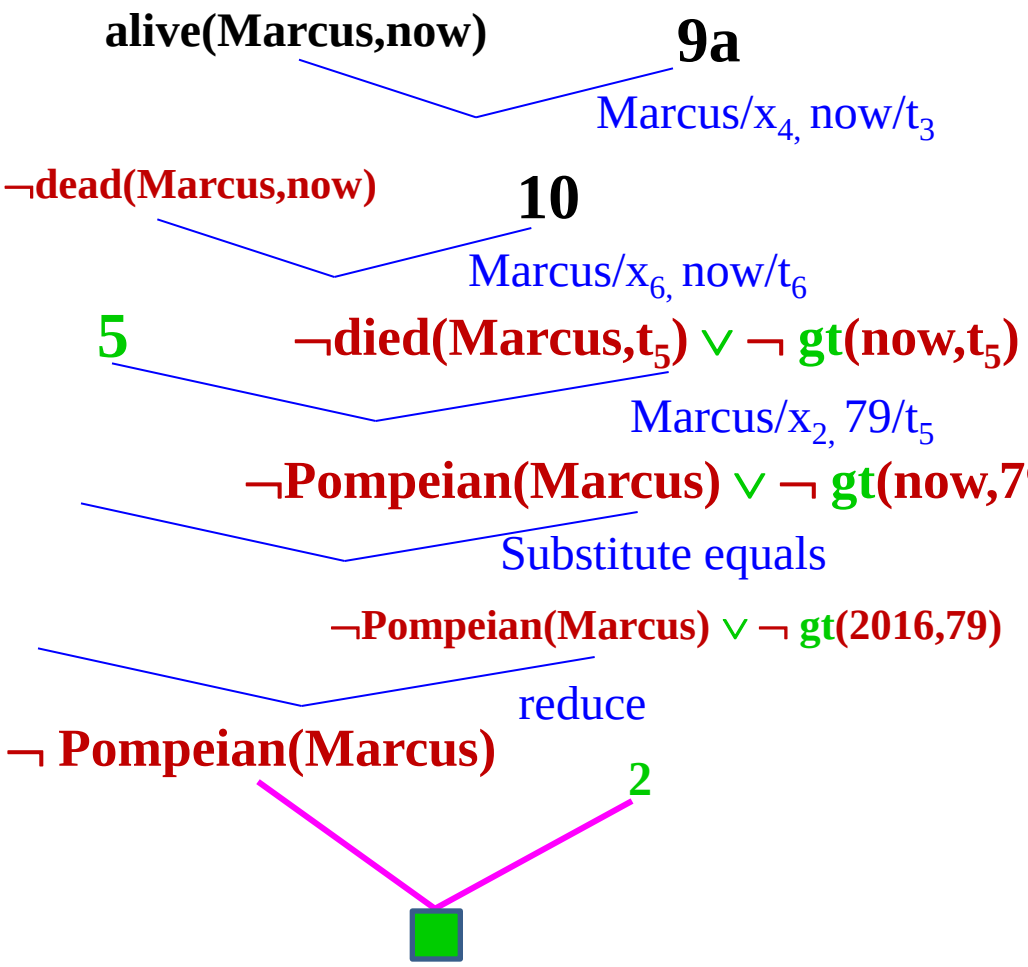
-

# An Example

- Now to detect that there is no contradiction we must discover that the only resolvents that can be generated have been generated before.
- In other words, although we can generate resolvents, we can generate no new ones.
- In its pure form, resolution requires all the knowledge it uses to be represented in the form of clauses.
- It is more efficient to represent certain kinds of info in the form of computable functions, computable predicates, & equality relationships

# An Example

**Prove: ¬ alive(Marcus,now)**




1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. $\neg man(x_1) \vee mortal(x_1)$
5. $\neg Pompeian(x_2) \vee died(x_2, 79)$
6. erupted(volcano,79)
7. $\neg\ mortal(x_3) \vee \neg\ born(x_3,t_1) \vee \neg gt(t_2-t_1,150) \vee dead(x_3,t_2)$
8. now=2016

9a. $\neg\ alive(x_4,t_3) \vee \neg dead(x_4,t_3)$

9b. $dead(x_5, t_4) \vee alive(x_5, t_4)$

10. $\neg died(x_6, t_5) \vee \neg\ gt(t_6, t_5) \vee dead(x_6,t_6)$

Using resolution with equality & reduce

# Two ways of generating new clauses in addition to the resolution rule

1. Substitution of one value for another to which it is equal
2. Reduction of computable predicates. If the predicate evaluates to FALSE, it can simply be dropped, since adding V FALSE to a disjunction cannot change its truth value.

   -If the predicates evaluates to TRUE, then the generated clause is a tautology & cannot lead to a contradiction

# Chapter Summary

- Logic: Propositional Logic & Predicate Logic
- Facts to Predicated logic (CNF)
- Computable functions & Predicates
- Resolution
- Unification