# KNOWLEDGE REPRESENTATION

## Chapter #3

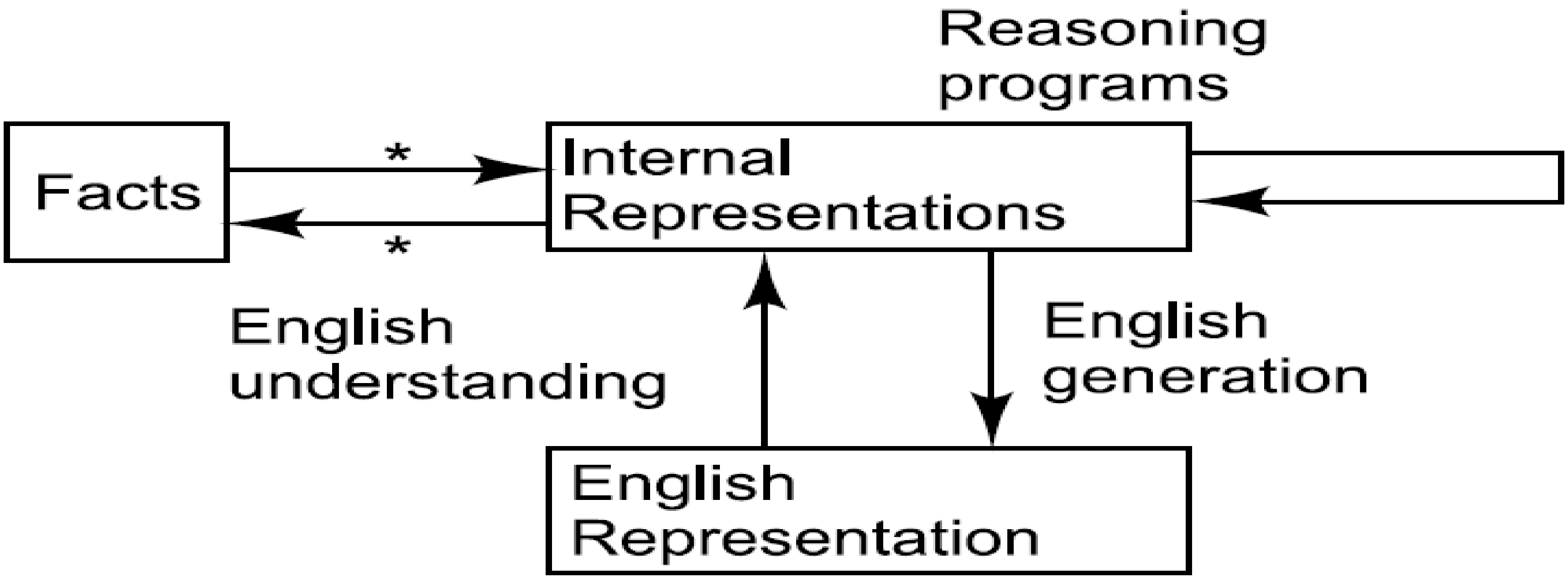
# Knowledge Representation

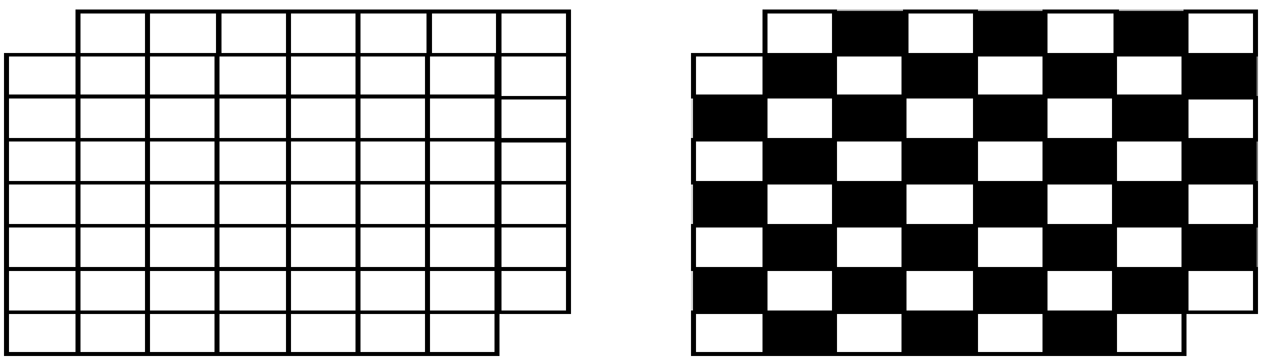
- For representing knowledge we deal with two different entities.

1. Facts : truths in some relevant world.
2. Representation of facts in some chosen formalization (such as rules)

- Structuring these entities:

1. Knowledge level
2. Symbol level


Reasoning
programs
Facts
*
*
Internal
Representations
English
understanding
English
generation
English
Representation



(a) (b)

Number of
black squares = 30

Number of
white squares = 32

(c)

# Approaches to Knowledge Representation

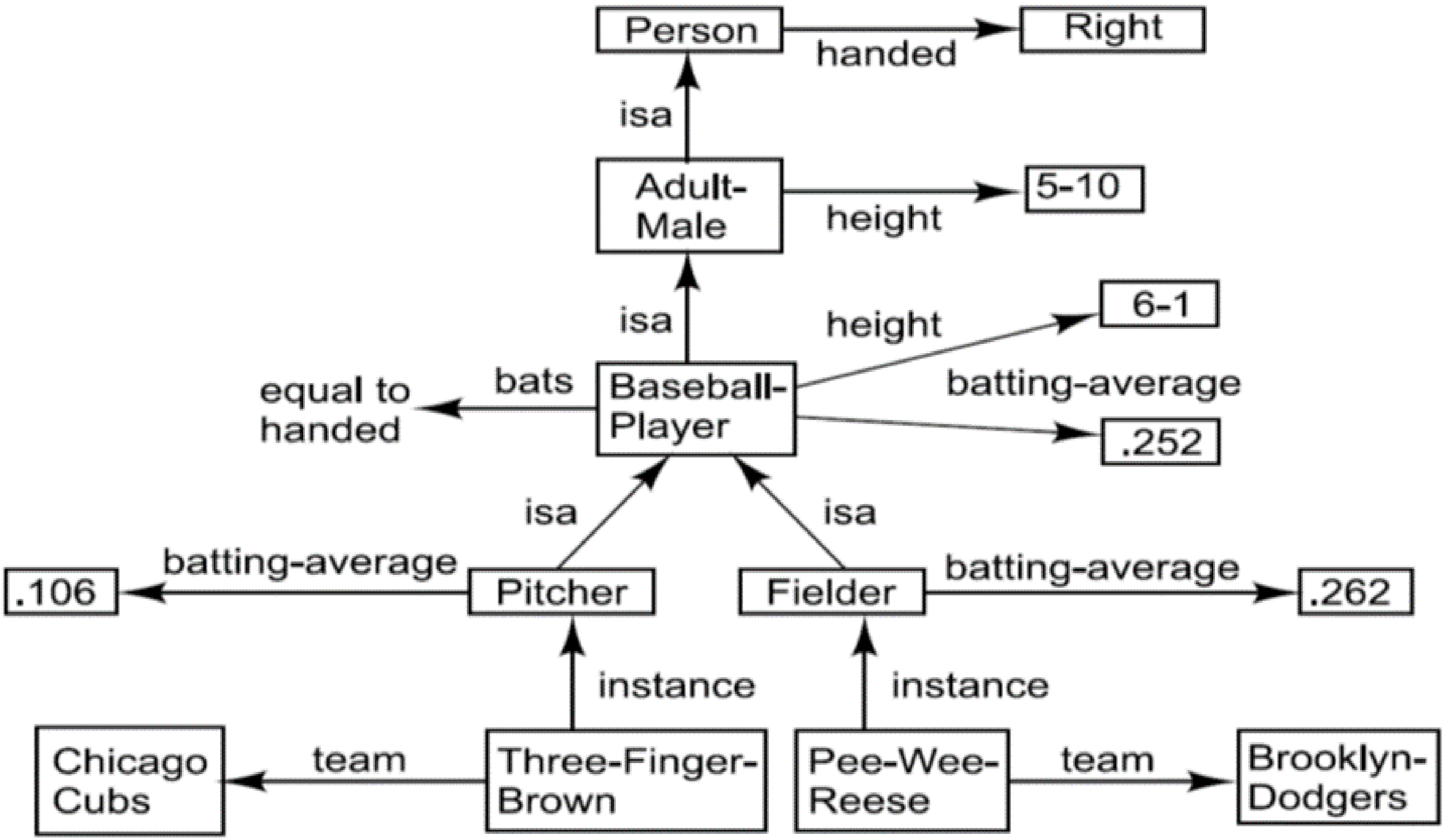
A good system for the representation of knowledge in Particular domain should possess the following properties:

- **Representational Adequacy**: the ability to represent all kind of knowledge.
- **Inferential Adequacy:** deriving new structure from old structure by manipulating the knowledge.
- **Inferential efficiency**
- **Acquisition Efficiency:** the ability to acquire new information easily

# Simple Relational Knowledge

| Player | Height | Weight | Bats-Throws |
|---|---|---|---|
| Hank Aaron | 6-0 | 180 | Right-Right |
| Willie Mays | 5-10 | 170 | Right-Right |
| Babe Ruth | 6-2 | 215 | Left-Left |
| Ted Williams | 6-3 | 205 | Left-Right |
| player_info('hank aaron', '6-0', 180,right-right). | | | |

# Inheritable Knowledge



# Inferential knowledge

$\forall x : Ball(x) \wedge Fly(x) \wedge Fair(x) \wedge Infield\text{-}Catchable\ (x) \wedge$
$Occupied\text{-}Base(First) \wedge Occupied\text{-}Base(Second) \wedge (Outs < 2) \wedge$
$\neg[Line\text{-}Drive(x) \vee Attempted\text{-}Bt,(x)]$
$\rightarrow Infield\text{-}Fly(x)$

$\forall x,y : Batter(x) \wedge batted(x, y) \wedge Infield\text{-}Fly(y) \rightarrow \mathrm{Out}(x)$

# Procedural Knowledge (Using LISP code to define a value)

```
Baseball-Player
    isa:               Adult-Male
    bats:              (lambda (x)
                           (prog ()
                                 L1
                             (cond ((caddr x) (return (caddr x)))
                                   (t (setq x (eval (cadr x)))
                                      (cond (x (go L1))
                                            (t (return nil)))))))
height:                6-1
batting-average:       .252
```

# Procedural knowledge as Rules

| | |
|---|---|
| If: | ninth inning, and |
| | score is close, and |
| | less than 2 outs, and |
| | first base is vacant, and |
| | batter is better hitter than next batter, |
| Then: | walk the batter. |

# Issues in knowledge representation

- **Are there any basic attributes of objects?**
- **Are there any basic relationships among objects?**
- **At what level should knowledge be represented?**
- **How should sets be represented?**
- **How should knowledge be accessed?**

# Issues

1. Important Attributes

There are two attributes that are of very general significance(instance, isa) . These attributes are very important because they support property inheritance.

2. Relationship among attributes

- Inverses $team(Pee\text{-}Wee\text{-}Reese,\ Brooklyn\text{-}Dodgers)$
- **Existence in an isa hierarchy:** generalization- specialization relationship are important for attributes as they are important for other concepts(eg: height;physical size; physical)

- **Techniques for reasoning about values:**

1. Info about type of value (height is in number measured in unit of length)
2. Constraints on the value(age of child is not greater than age of parent)
3. Rules for computing the value when it is needed

- **Single valued attributes:** that take only a unique value.( baseball player at any one time , have only single height and be member of only one team)
- If some different value is asserted , then one of the two things happen:
- Either a change has occurred in world or there is now a contradiction in knowledge base

# Choosing granularity of representation

At what level of detail should the knowledge is represented?

**Suppose we are interested in the following fact :**

John spotted Sue.

**We could represent this as**

*spotted(agent(John),*

*object(Sue))*

**Questions :**

**Who spotted Sue?**

**Did John see Sue?**

**We could add facts, e.g.:**

$spotted(x, y) \longrightarrow saw(x, y)$

**An alternative representation:**

*saw(agent(John),*

*object(Sue),*

*timespan(briefly))*

**Mary is Sue's cousin.**

- $\text{Mary} = daughter(brother(mother(Sue)))$
- $\text{Mary} = daughter(sister(mother(Sue)))$
- $\text{Mary} = daughter(brother(father(Sue)))$
- $\text{Mary} = daughter(sister(father(Sue)))$

**An alternative:**

- $\text{Mary} = child(sibling(parent(Sue)))$

# Representing Sets of objects

There are two ways to state a definition of sets and its elements:

1) Extensional definition: In this we list all the members
2) Intensional definition: Provide a rule that, when a particular object is evaluated , returns true or false depending on whether the object is in the set or not.

Problem: Planet on which people live.
Extensional :*{Earth}*

Intensional:

$$\{x : sun\text{-}planet(x) \wedge human\text{-}inhabited(x)\}$$

$$\{x : sun\text{-}planet\{x) \ ?\ nth\text{-}farthest\text{-}fmm\text{-}sun(x, 3)\}$$
$$\{x : sun\text{-}planet(x) \ ?\ nth\text{-}biggest(x, 5)\}$$

Intensional representation have two important properties:

1) They can be used to describe infinite set.
2) Allow them to depend on parameters that can change such as time.

# Finding the right structure as needed

Issue is to locate appropriate knowledge structure that have been stored in memory.

In order to access a correct structure for describing a particular situation, it is necessary to solve following problems:

1)How to perform an initial selection of the most appropriate structure.

2) How to fill in appropriate details from the current situation

3)How to find a better structure if initially chosen turns out to be not appropriate.

4)What to do if none of the available structure is appropriate

5) When to create and remember a new structure

# Selecting an Initial Structure

- Index structures directly by English words that can be used to describe them(for example verb). Problem:

  - John flew to New York.

    * He rode in a plane from one place to another.

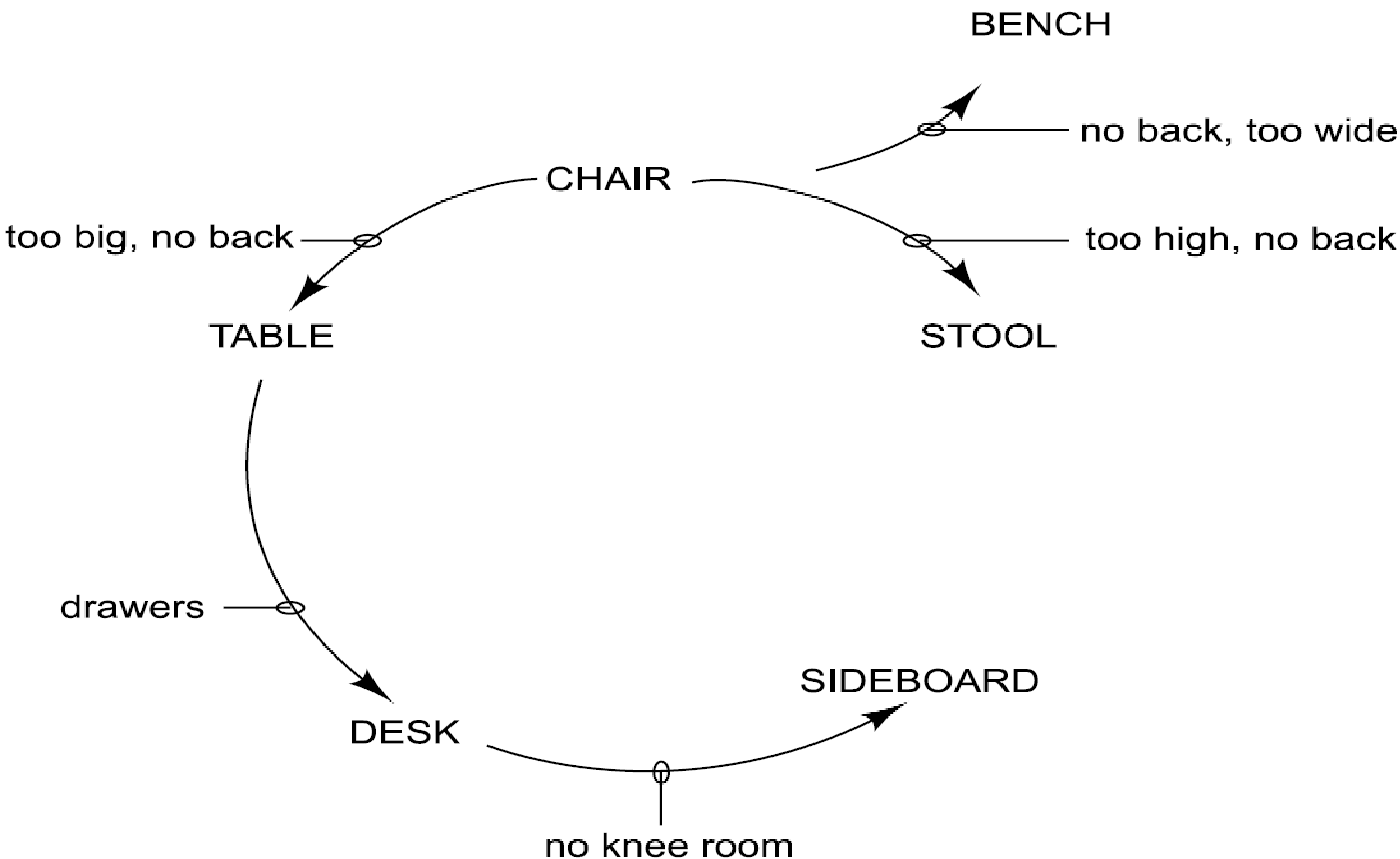
  - John flew a kite.

    * He held a kite hat was up in the air.

  - John flew down the street.

    * He moved very rapidly.

- Each major concept as pointer to all the structure (like bill pointer is used to point two scripts such as restaurant and shopping)
- Use one major clue in problem description and use it to select an initial structure (problem is that sometimes the major clue is not easily identifiable, other is that which clues are going to be important and which are not)


BENCH
no back, too wide
CHAIR
too big, no back
too high, no back
TABLE
STOOL
drawers
SIDEBOARD
DESK
no knee room

- **Representing the state**

  **- Store all the facts at each node.**

  *** Problem : a lot of facts get represented a lot of times.**

  ***above (Ceiling, Floor)***

  **- Store a representation of the changes.**

  **- Modify the state but record how to undo.**

- **Computing the new state : Frame axioms**

$$color(x,y,\ s_1) \wedge move(x,\ s_1,S_2) \rightarrow color(x,y,\ S_2)$$

# Using Predicate Logic

We begins with exploring one particular way of representing facts- the language of logic.

It is a powerful way of deriving new knowledge from old

**Propositional logic**

In this we represent real world facts as logical propositions

# Some Simple Facts in Propositional Logic

It is raining.

$RAINING$

It is sunny.

$SUNNY$

It is windy.

$WINDY$

If it is raining, then it is not sunny.

$RAINING \rightarrow \neg SUNNY$

# Limitations of Propositional Logic

**Socrates is a man.**

***SOCRATESMAN***

**Plato is a man.**

***PLATOMAN***

**Better representations :**

***MAN*(*SOCRATES*)**

***MAN*(*PLATO*)**

**All men are mortal.** **MORTALMAN**

**Better representation :**

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

# A Predicate Logic Example

1. Marcus was a man.

2. Marcus was a Pompeian.

3. All Pompeian were Romans.

4. Caesar was a ruler.

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

1. Everyone is loyal to someone.

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

**Prove: Marcus tried to assassinate Caesar.**

# A Predicate Logic Example

1. Marcus was a man.
   $man(Marcus)$
2. Marcus was a Pompeian.
   $Pompeian(Marcus)$
3. All Pompeians 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.
   $\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
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.
   $\forall x : man(x) \rightarrow person(x)$

# Representing instance and isa relationship

The predicate instance is a binary one, whose first argument is a object and second argument is a class to which object belongs.

The predicate instance is used to show the relation among 2 classes.

# Three Ways of Representing Class Membership

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\forall x : Pompeian(x) \rightarrow Roman(x)$
4. $ruler(Caesar)$
5. $\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $instance(Marcus, Pompeian)$
3. $\forall x : instance(x, Pompeian) \rightarrow instance(x, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $\text{instance}(Marcus, Pompeian)$
3. $isa(Pompeian, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
6. $\forall x : \forall y : \forall z : instance(x, y) \wedge \text{isa}(y, z) \rightarrow instance(x, z)$

# Overriding Defaults

**Paulus is Pompeian**

**Paulus is neither loyal to caesar nor hated caesar**

**Suppose we add:**

$$Pompeian(Paulus)$$
$$\neg\ [loyalto(Paulus,\ Caesar) \vee hate(Paulus, Caesar)]$$

**But now we have a problem with 5:**

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

**So we need to change it to :**

$$\forall x : Roman(x) \wedge \neg eq(x, Paulus) \rightarrow$$

$$loyalto(x, Caesar) \vee hate(x, Caesar)$$

# Another Predicate Logic Example

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.
8. Alive means not dead.

# A Set of Facts about Marcus

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\forall x : man(x) \rightarrow mortal(x)$
5. $\forall : Pompeian(x) \rightarrow died(x, 79)$
6. $erupted(yolcano, 79)$
7. $\forall_x : \forall t_1 : \forall t_2 : mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \rightarrow dead(x, t_2)$
8. $now = 1991$
9. $\forall x : \forall t$: [alive$(x, t) \rightarrow \neg dead(x, t)] \wedge [\neg dead(x, t) \rightarrow alive(x, t)]$
10. $\forall x : \forall t_1 : \forall t_2 : died(x, t_1) \wedge gt(t_2, t_1) \rightarrow dead(x, t_2)$

5. Erupted(volcano, 79)/\ $\forall : Pompeian(x) \rightarrow died(x, 79)$

# One Way of Proving That Marcus Is Dead

$$
\begin{array}{ll}
\neg alive(Marcus,\ now) & \\
\quad\uparrow & (9,\ \text{substitution}) \\
dead(Marcus,\ now) & \\
\quad\uparrow & (10,\ \text{substitution}) \\
died(Marcus,\ t_1) \wedge gt(now,\ t_1) & \\
\quad\uparrow & (5,\ \text{substitution}) \\
Pompeian(Marcus) \wedge gt(now,\ 79) & \\
\quad\uparrow & (2) \\
gt(now,\ 79) & \\
\quad\uparrow & (8,\ \text{substitute equals}) \\
gt(1991, 79) & \\
\quad\uparrow & (\text{compute gt}) \\
nil &
\end{array}
$$

# Another Way of Proving That Marcus Is Dead

$$
\begin{array}{ll}
\neg alive(Marcus,\ now) & \\
\quad\uparrow & (9,\ \text{substitution}) \\
dead(Marcus,\ now) & \\
\quad\uparrow & (7,\ \text{substitution}) \\
mortal(Marcus)\ \wedge & \\
born(Marcus,\ t_1)\ \wedge & \\
gt(now - t_1,\ 150) & \\
\quad\uparrow & (4,\ \text{substitution}) \\
man(Marcus)\ \wedge & \\
born(Marcus,\ t_1)\ \wedge & \\
gt(now - t_1,\ 150) & \\
\quad\uparrow & (1) \\
born(Marcus,\ t_1)\ \wedge & \\
gt(now - t_1,\ 150) & \\
\quad\uparrow & (3) \\
gt(now - 40, 150) & \\
\quad\uparrow & (8) \\
gt(1991 - 40, 150) & \\
\quad\uparrow & (\text{compute minus}) \\
gt(1951, 150) & \\
\quad\uparrow & (\text{compute gt}) \\
nil &
\end{array}
$$

# Conversion to Clause Form

**Problem :**

$$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow$$
$$[hate(x,Caesar) \vee (\forall y : \exists z : hate(y, z) \rightarrow thinkcrazy(x, y))]$$

**Solution :**

- **Flatten**
- **Separate out Quantifiers**

**Conjunctive Normal Form :**

$$\neg Roman(x) \wedge \neg know(x, Marcus) \vee$$
$$hate(x, Caesar) \vee \neg hate(y, z) \vee thinkcrazy(x, z)$$

**Clause Form :**

- **Conjunctive Normal Form**
- **No instances of** $\wedge$

# Algorithm : Convert to Clause Form

1. Eliminate $\rightarrow$, using: $a \rightarrow b = \neg a \vee b$.
2. Reduce the scope of each $\neg$ to a single term, using:
   - $\neg(\neg p) = p$
   - deMorgan's laws: $\neg(a \wedge b) = \neg a \vee \neg b$
     $\neg(a \vee b) = \neg a \wedge \neg b$
   - $\neg \forall x : P(x) = \exists x : \neg P(x)$
   - $\neg \exists x : P(x) = \forall x : \neg P(x)$
3. Standardize variables.
4. Move all quantifiers to the left of the formula without changing their relative order.
5. Eliminate existential quantifiers by inserting Skolem functions.
6. Drop the prefix.
7. Convert the matrix into a conjunction of disjuncts, using associativity and distributivity.
8. Create a separate clause for each conjunct.
9. Standardize apart the variables in the set of clauses generated in step 8, using the fact that

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

# Examples of Conversion to Clause Form

**Example :**

$$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow [hate(x, Caesar) \vee (\forall y : \exists z : hate(y, z) \rightarrow thinkcrazy(x, y))]$$

**1 Eliminate $\rightarrow$**

$$\forall x : \neg[Roman(x) \wedge know< x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y : \neg(\exists z : hate(y, z)) \vee thinkcrazy(x,y))]$$

**2 Reduce scope of $\neg$**

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

**3 Standardize Variables.**

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

would be converted to

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

# Examples of Conversion to Clause Form

**4 Move quantifiers.**

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

**5 Eliminate existential quantifiers.**

$\exists y : President(y)$

**will be converted to**

$President(S1)$

**while**

$\forall x : \exists y : father\text{-}of(y,x)$

**will be converted to**

$\forall x : father\text{-}of(S2(x),x))$

**6 Drop the prefix.**

$$[\neg Roman(x) \vee \neg know(x, Marcus)] \vee$$
$$[hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkcrazy\{x, y))]$$

**7 Convert to a conjunction of disjuncts.**

$$\neg Roman(x) \vee \neg know(x, Marcus) \vee$$
$$hate(x, Caesar) \vee \neg hate(y, z) \vee thinkcrazy(x, y)$$

# Examples of Conversion to Clause Form

**The Formula**

$$(winter \wedge wearingboots) \vee (summer \wedge wearingsandals)$$

becomes, after one application of the rule

$$[winter \vee (summer \wedge wearingsandals)]$$
$$\wedge\ [wearingboots \vee (summer \wedge wearingsandals)]$$

**becomes**

$$[winter \vee (summer \wedge wearingsandals)]$$
$$\wedge\ [wearingboots \vee (summer \wedge wearingsandals)]$$

**and then becomes**

$$(winter \vee summer) \wedge$$
$$(\mathrm{winter} \vee wearingsandals) \wedge$$
$$(wearingboots \vee summer) \wedge$$
$$(wearingboots \vee wearingsandals)$$

# The Basis of Resolution

**Resolution procedure is a simple iterative process : at each step , two Clauses (parent clauses) are compared, yielding a new clause that has been inferred from them.**

$$winter \lor summer$$
$$\neg winter \lor cold$$

**becomes**

$$summer \lor cold$$

# Algorithm : Propositional Resolution

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

2. Negate *P* and 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. The resulting clause, called the *resolvent,* will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals *L* and ¬*L* such that one of the parent clauses contains *L* and the other contains ¬*L,* then select one such pair and eliminate both *L* and ¬*L* 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.

# A Few Facts in Propositional Logic

Given Axioms

$P$

$(P \wedge Q) \rightarrow R$

$(S \vee T) \rightarrow Q$

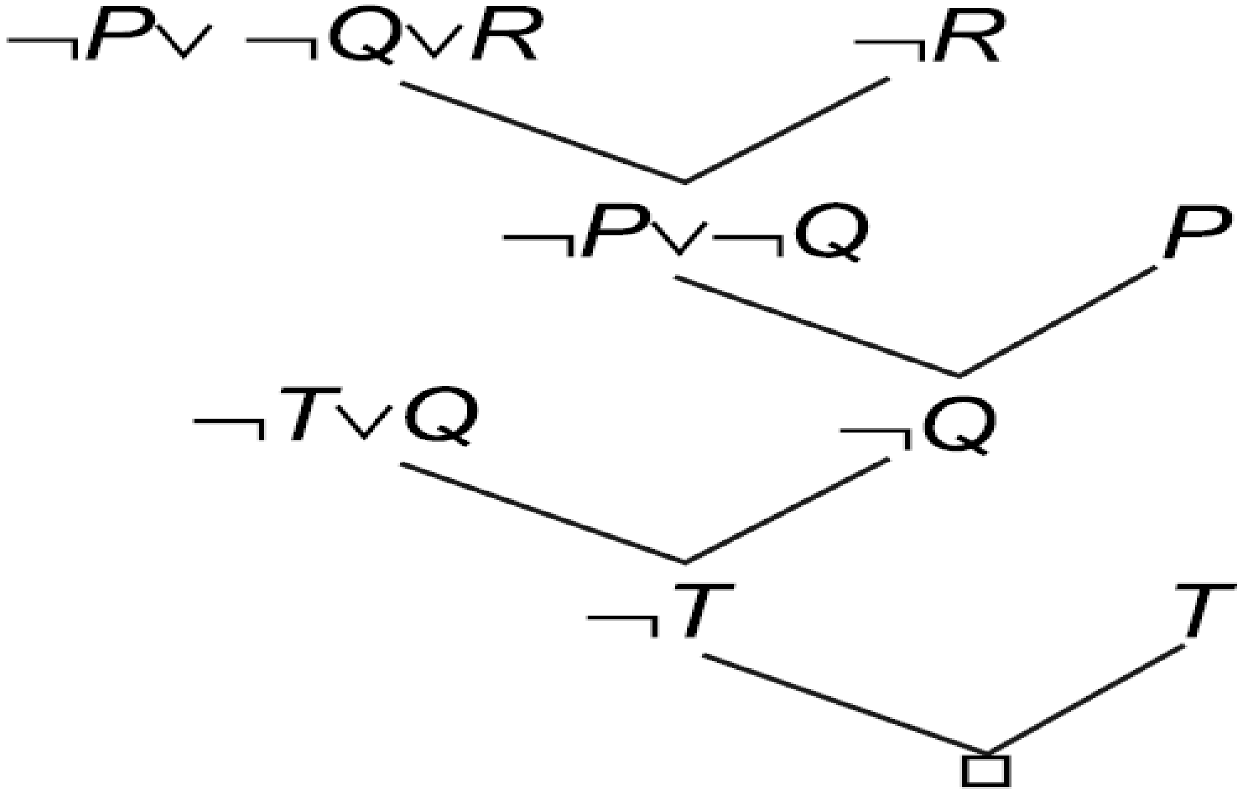
$T$

# CNF Form

Converted to Clause Form

| | |
|---|---|
| $P$ | (1) |
| $\neg P \vee \neg Q \vee R$ | (2) |
| $\neg S \vee Q$ | (3) |
| $\neg T \vee Q$ | (4) |
| $T$ | (5) |

# Resolution in Propositional Logic
## Prove : R is true



# Unification

Unification is a matching procedure that compares two literals and discover whether there exists a set of substitutions that makes them identical.

To attempt to unify two literals we first check if their initial predicate symbols are same.

# Unification

$Q(x)$
$P(y)$ $\longrightarrow$ FAIL

$P(x)$
$P(y)$ $\longrightarrow x/y$

$P(Marcus)$
$P(y)$ $\longrightarrow$ Marcus/$y$

$P(Marcus)$
$P(Julius)$ $\longrightarrow$ FAIL

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

$P(y, y)$

# Finding General Substitutions

**Given :**

$$hate(x,\ y)$$
$$hate(Marcus,\ z)$$

**We could produce :**

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

# Algorithm : Unify (L1, L2)

1. If *L*1 or *L*2 are both variables or constants, 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*I and *L*2 have a different number of arguments, then return {FAIL}.

4. Set *SUBST* to NIL.

5. For *i* ← 1 to number of arguments in *L*1:

   (a) Call Unify with the /th argument of *L*1 and the *i*th argument of *L*2, 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 *L*1 and *L*2*.*

      (ii) *SUBST :* = APPEND(S, *SUBST*)*.*

6. Return *SUBST.*

# Why Do the Occur Check?

**Example :**

$$f(x, x)$$

$$f(g(x), g(x))$$

# Resolution in Predicate Logic

**Example :**

1. $man(Marcus)$
2. $\neg man(x_1) \vee mortal(x_1)$

**Yield the substitution :**

**Marcus/$x_1$**

**So it does not yield the resolvent :**

**mortal/$x_1$**

**It does yield :**

**mortal(Marcus)**

# 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 $\neg T2$ 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. If there is more than one pair of complimentary literals, only one pair shold 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.

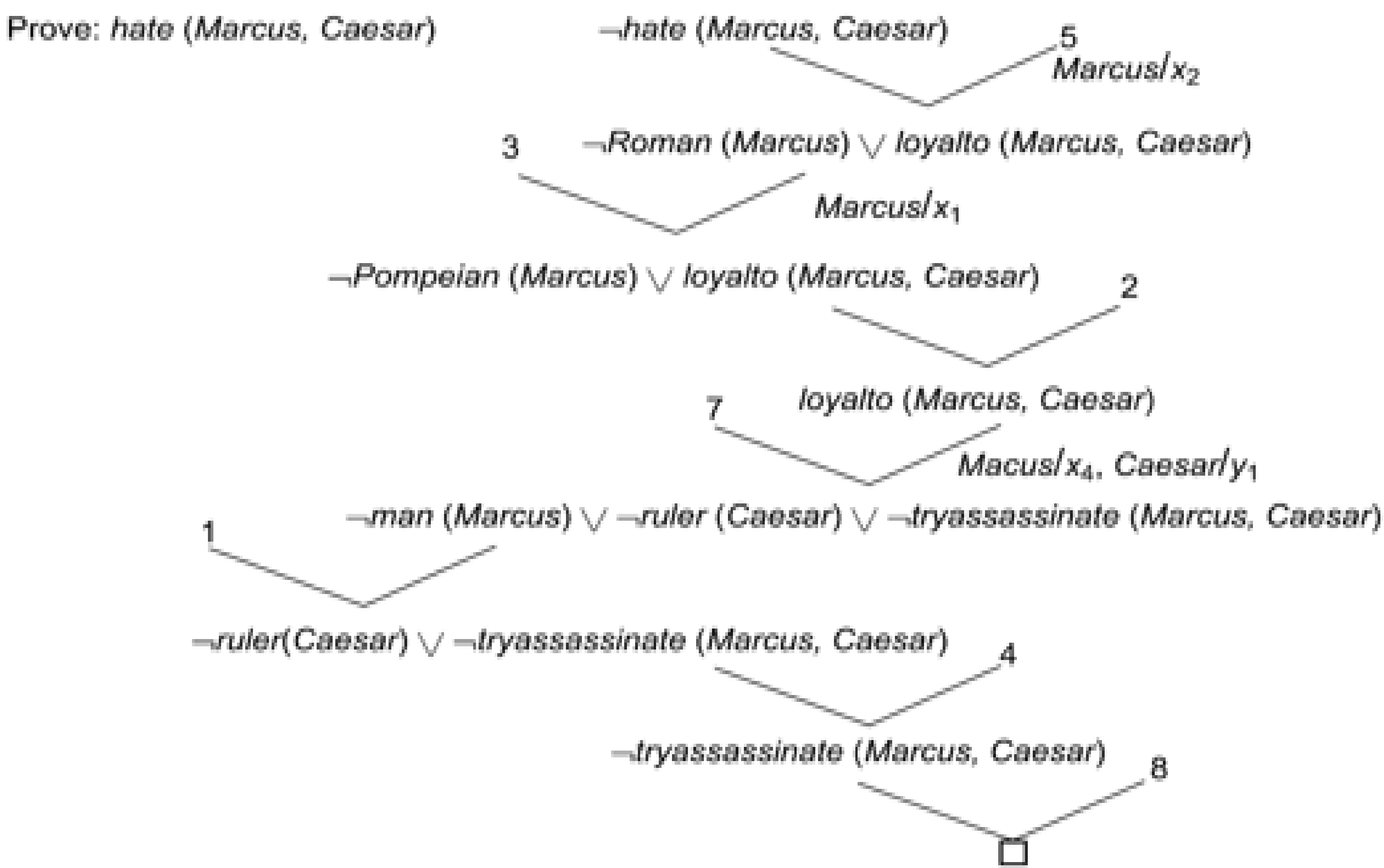
# A Resolution Proof

Axioms in clause form:

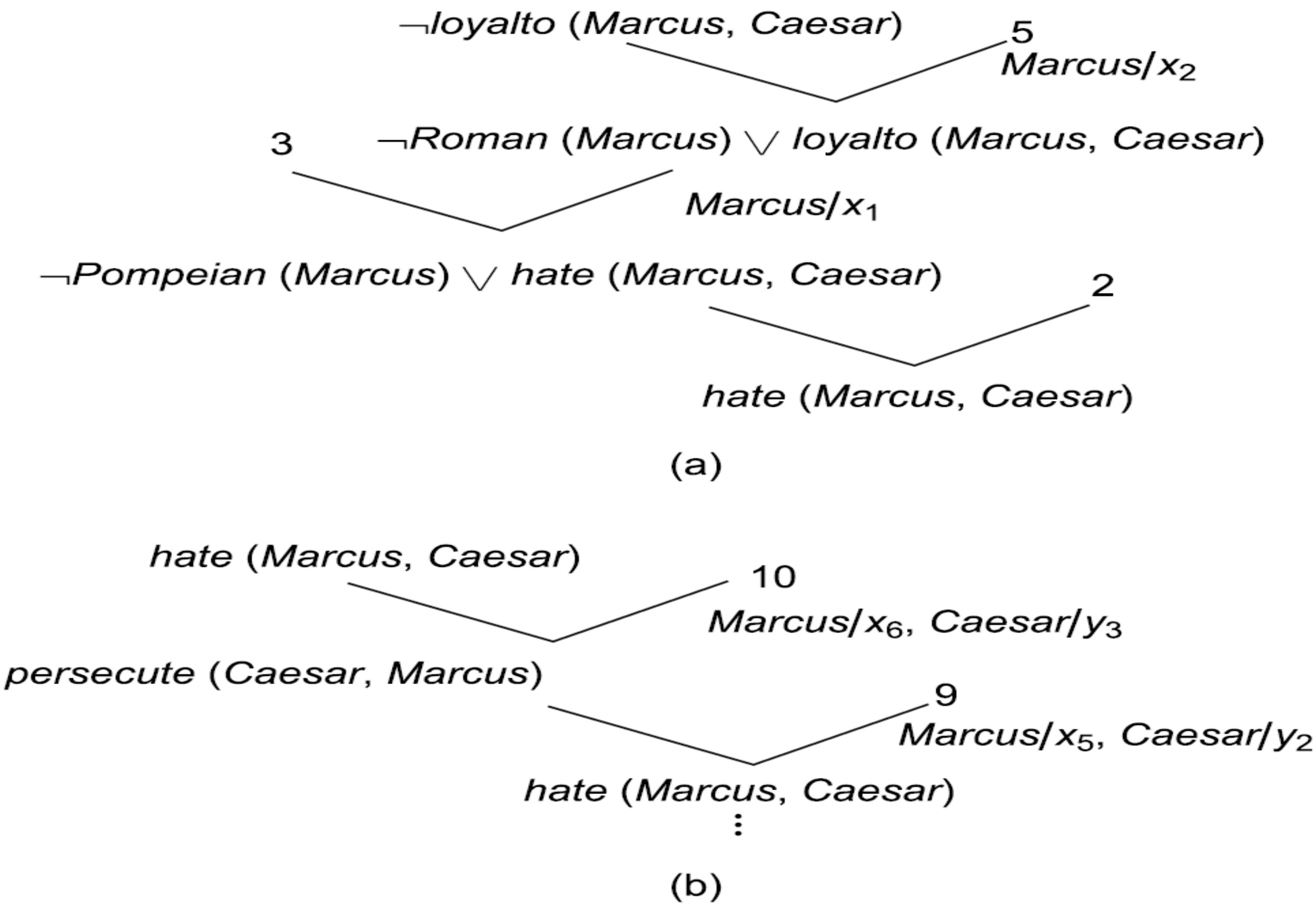
1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f1(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus,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

(a)

Prove: hate(Marcus, Caesar)


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
□

# An Unsuccessful Attempt at Resolution



# The Need to Standardize Variables

Given:

1. $\neg father(x, y) \vee \neg woman(x)$

   (i.e., $father(x, y) \rightarrow \neg woman(x)$)
2. $\neg mother(x, y) \vee woman(x)$

   (i.e., $mother(x, y) \rightarrow woman(x)$)
3. $mother(Chris, Mary)$
4. $father(Chris, Bill)$

1 2

$\neg father\ (x,y)\ \neg Mother,\ (x,y)$ 3

$Chris/x,\ Mary/y$

$\neg father\ (Chris,\ Mary)$

# Using Resolution with Equality and Reduce

Axioms in clause form:

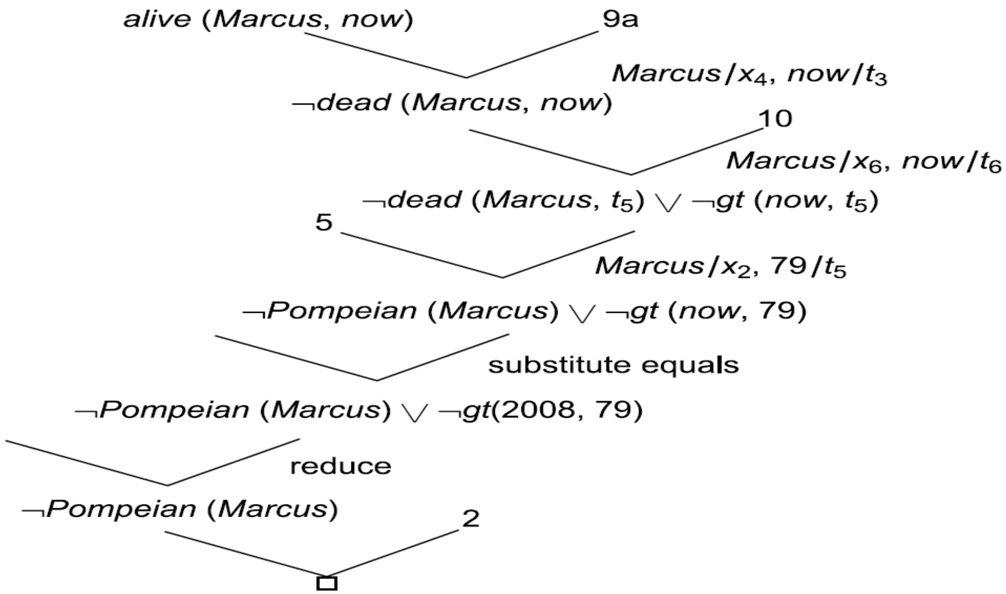
1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $horn(Marcus, 40)$
4. $\neg man(x_1) \vee mortal(x_1)$
5. $\neg Pompeian(x_2) \vee died(x_2, 79)$
6. $erupted(volcano, 79)$
7. $\neg morta(x_3) \vee \neg born(x_3, t_1) \vee \neg gt(t_2 - t_1, 150) \vee dead(x_3, t_2)$
8. $now = 2008$

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

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

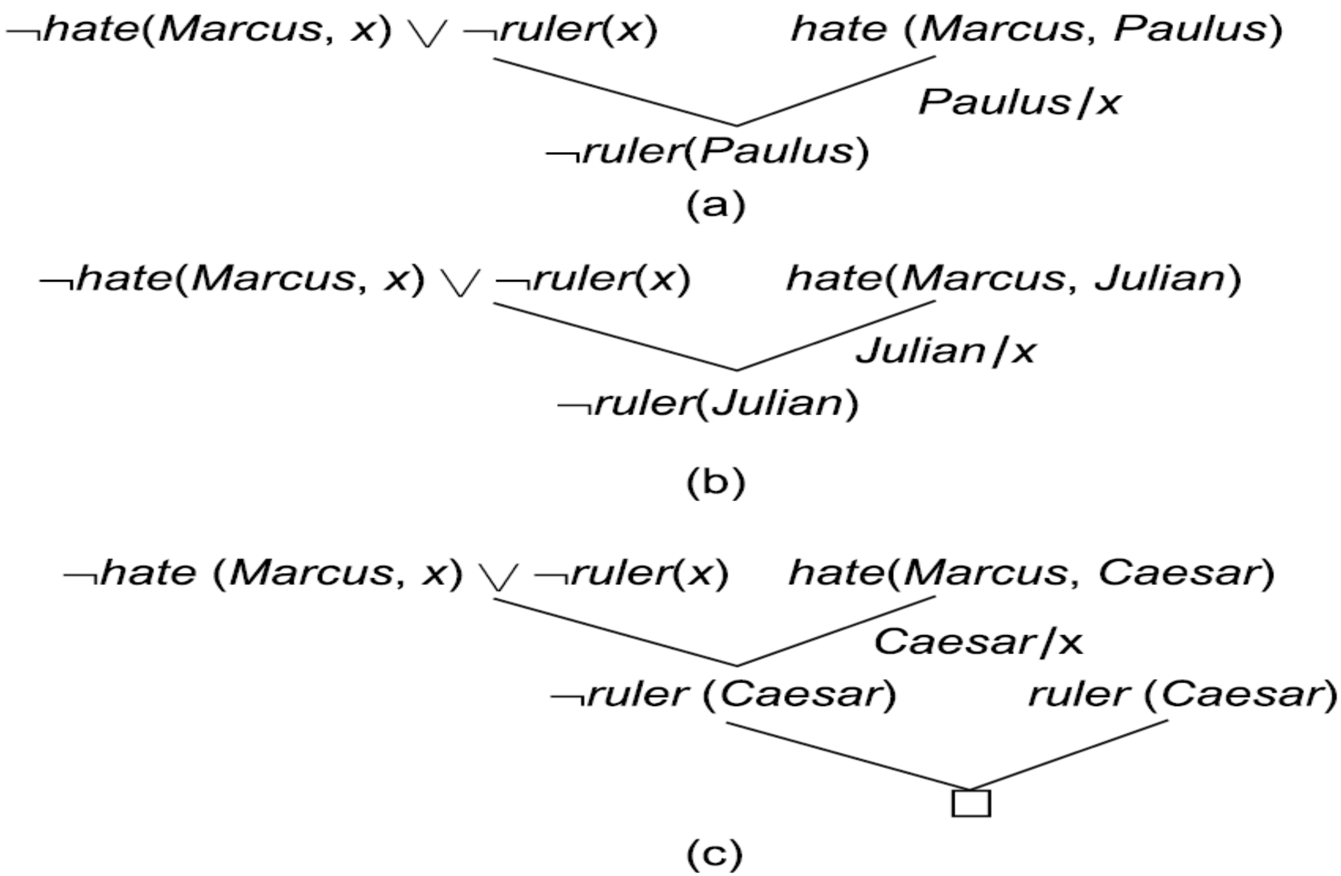
10. $\neg died(x_6\ t_5) \vee \neg gt(t_6, t_5) \vee dead(x_6, t_6)$

Prove: $\neg alive(Marcus, now)$

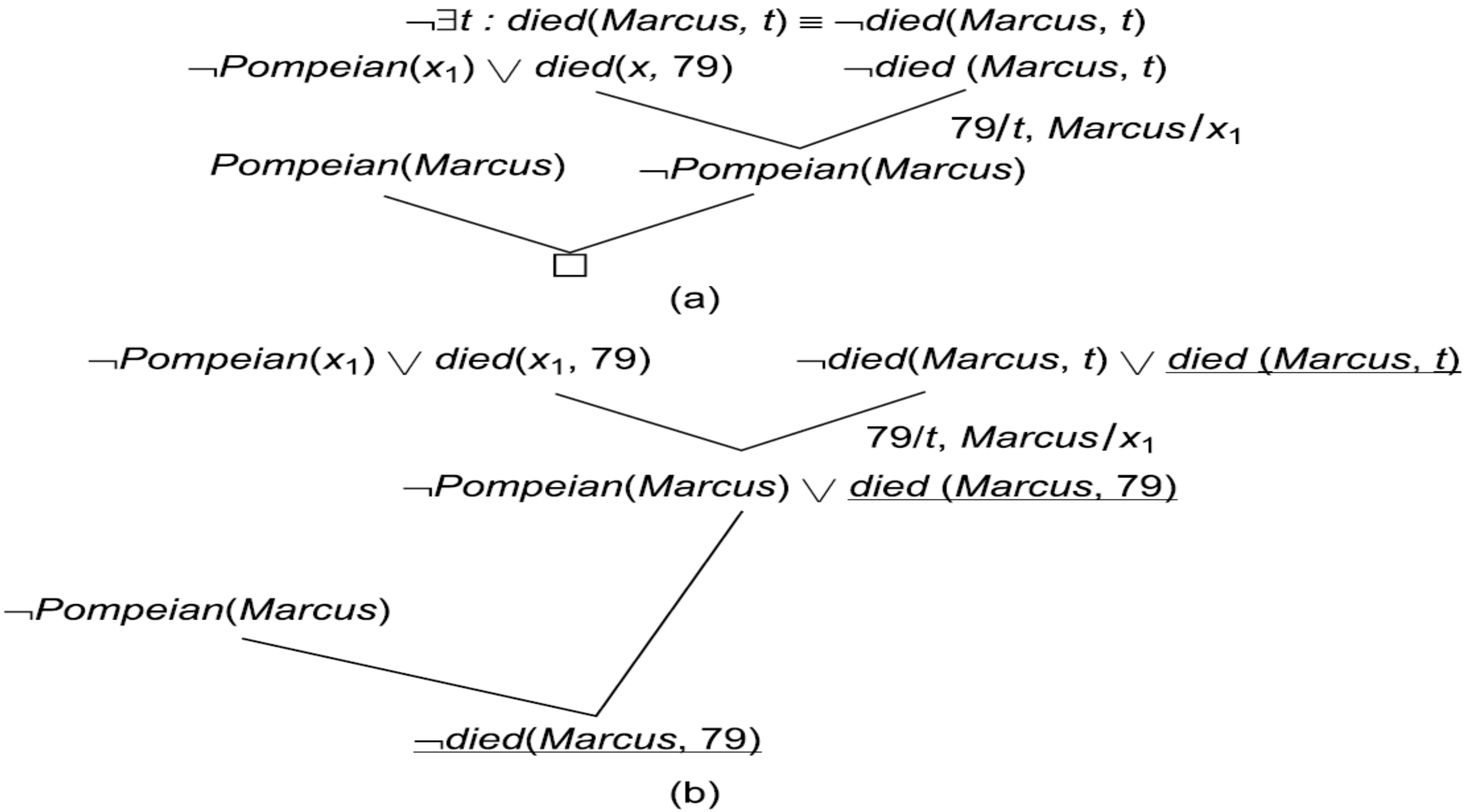


# Trying Several Substitutions

| | |
|---|---|
| Prove: | $\exists x : hate(Marcus, x) \wedge ruler(x)$ |
| (negate): | $\neg \exists x : hate(Marcus, x) \wedge ruler(x)$ |
| (clausify): | $\neg hate(Marcus, x) \vee \neg ruler(x)$ |



# Answers Extraction Using Resolution



# The Need to Change Representations

"What happened in 79 A.D.?"

$$\exists x : event(x, 79)$$

But we have

erupted(volcano, 79)

$\neg event(x,79) \vee \underline{event(x, 79)}$ $\quad$ $event\ (erupted\ (volcano),79)$

$erupted\ (volcano)/x$

$\underline{event\ (erupted\ (volcano),79)}$

# Unification Examples

1. $f(Marcus)$
   $f(Caesar)$

2. $f(x)$
   $f(g(y))$

3. $f(Marcus, g(x, y))$
   $f(x, g(Caesar, Marcus))$

# Resolution Example

- John likes all kind of food.
- Apples are food.
- Chicken is food.
- Anything anyone eats and isn’t killed by is food.
- Bill eats peanuts and is still alive.
- Sue eats everything Bill eats.

# Resolution Example

- The members of the Elm St. Bridge Club are Joe, Sally, Bill and Ellen.
- Joe is married to Sally.
- Bill is Ellen’s brother.
- The spouse of every married person in the club is also in the club.
- The last meeting of the club was at Joe’s house.

# Resolution Example

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

# Order of Substitutions

$$loves(father\{a), \; a)$$

$$\neg loves(y, \; x) \vee loves(x, \; y)$$

# A Problem

## Given :

$$\forall x, y, z : gt(x, y) \wedge gt(y, z) \rightarrow gt(x, z)$$
$$\forall a, b : succ(a, b) \rightarrow gt(a, b)$$
$$\forall x : \neg gt(x, x)$$

## Prove :

$$gt(5, 2)$$

## What's wrong with :

$\neg gt(5, 2)$ $\qquad$ $\neg gt(x, y) \vee \neg gt(y, z) \vee gt(x, z)$

$5/x,\ 2/z$

$\neg gt(5, y) \vee \neg gt(y, 2)$ $\qquad$ $\neg succ(a, b) \vee gt(a, b)$

$y/a,\ 2/b$

$\neg gt(5, y) \vee \neg succ(y, 2)$ $\qquad$ $\neg gt(x, y) \vee \neg gt(y, z) \vee gt(x, z)$

$5/x,\ y/z$

$\neg gt(5, y) \vee \neg gt(y, y) \vee \neg succ(y, 2)$

T

# The Need for the Occur Check

**Unify:**

$$p(x, f(x))$$

$$p(f(a), a)$$

# KNOWLWDEGE REPRESENTAION USING RULES

# Procedural vs Declarative Knowledge

**Consider the knowledge base :**

$$man(Marcus)$$
$$man(Caesar)$$
$$person(Cleopatra)$$
$$\forall x : man(x) \rightarrow person(x)$$

**Suppose we want to answer the question**

$$\exists y : person(y)$$

**We could answer with any one of :**

$$y = Marcus$$
$$y = Caesar$$
$$y = Cleopatra$$

**Now consider an alternative KB :**

$$man(Marcus)$$
$$man(Caesar)$$
$$\forall x : man(x) \rightarrow person(x)$$
$$person(Cleopatra)$$

# PROLOG

A PROLOG program is composed of a set of Horn clauses.

A Horn Clause is a clause that has at most one positive literal.

Examples :

$p, \neg p \vee q,$

$r \rightarrow s \quad \neg r \vee s$

# A Declarative and a Procedural Representation

$\forall x : pet(x) \wedge small(x) \rightarrow apartmentpet(x)$

$\forall x : cat(x) \vee dog(x) \rightarrow pet(x)$

$\forall x : poodle(x) \rightarrow dog(x) \wedge small(x)$

$poodle(ftujfy)$

**A Representation in Logic**

```
apartmentpet(X) :- pet(X), small(X).
pet(X) :- cat(X).
pet(X) :- dog(X).
dog(X) :- poodle (X) .
small(X) :- poodle(X).
poodle(fluffy).
```

**A Representation in PROLOG**

# Answering Questions in PROLOG

```
apartmentpet(X) :- pet(X), small(X).
pet(X) :- cat(X).
pet(X) :- dog(X).
dog(X) :- poodle (X) .
small(X) :- poodle(X).
poodle(fluffy).

?- apartmentpet(X).

?- cat(fluffy).

?- cat (mittens)
```

# A Sample of the Rules for Solving the 8-Puzzle

Assume the areas of the tray are numbered:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Square 1 empty and Square 2 contains tile $n \rightarrow$
Square 2 empty and Square 1 contains tile $n$
Square 1 empty and Square 4 contains tile $n \rightarrow$
Square 4 empty and Square 1 contains tile $n$
Square 2 empty and Square 1 contains tile $n \rightarrow$
Square 1 empty and Square 2 contains tile $n$

⋮

An Examples :

Start

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

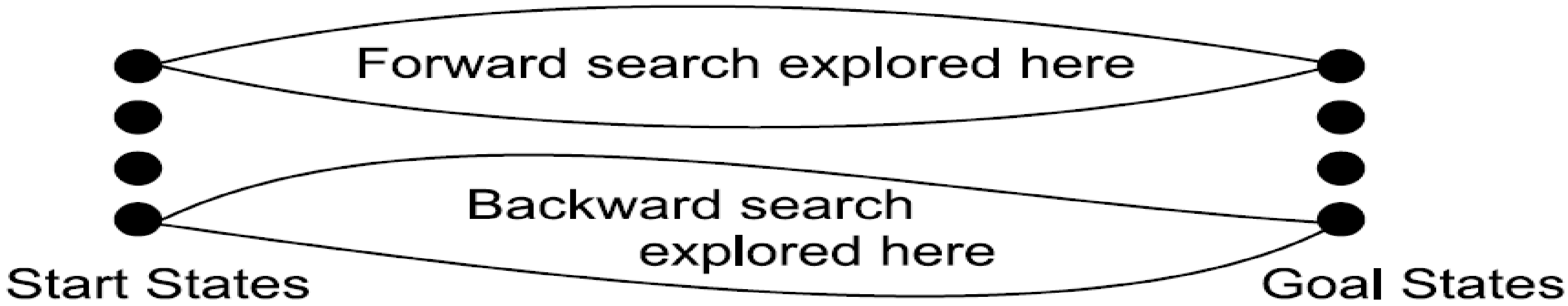
Goal

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

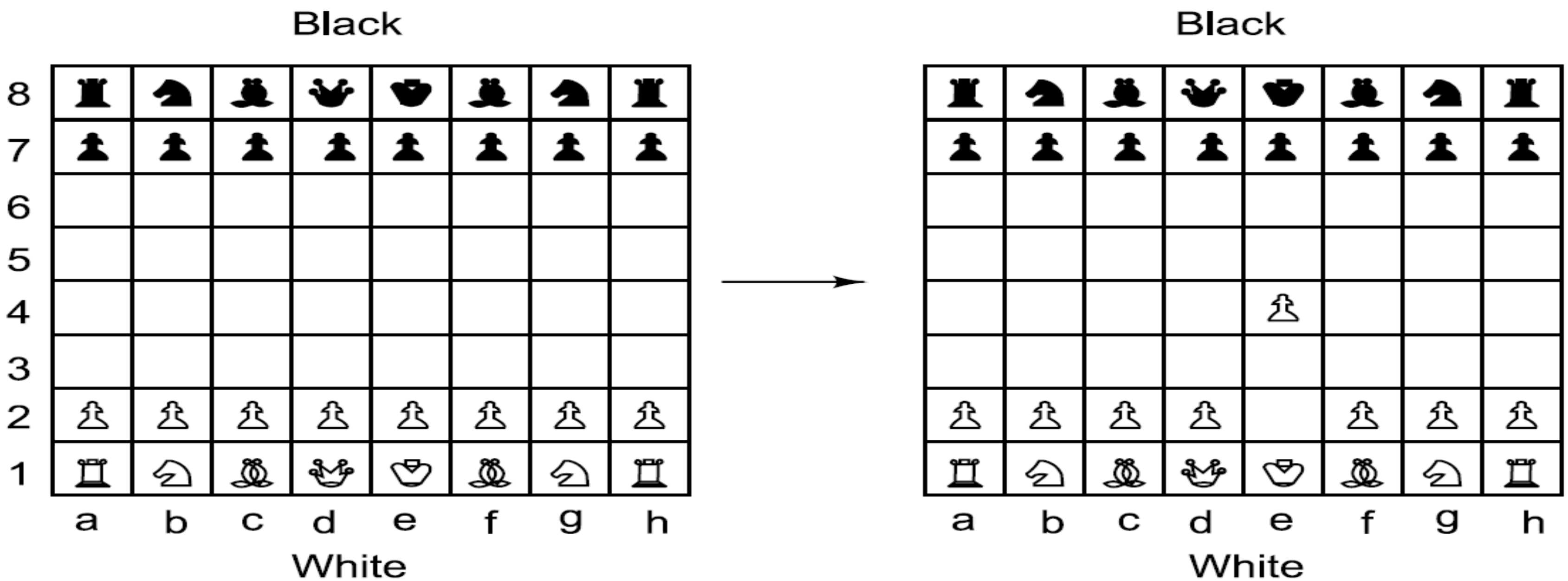
# Forward vs Backward Reasoning

- ❖ **Number of start and goal states.**
- ❖ **Branching factor in each direction.**
- ❖ **Need to justify reasoning.**
- ❖ **Triggers for problem solving.**

# A Bad Use of Heuristic Bidirectional Search



# Representation Affects Matching




White pawn at
Square(file e, rank 2)
AND
Square(file e, rank 3)
is empty
AND
Square(file e, rank 4)
is empty

→

move pawn from
Square(file e, rank 2)
to Square(file e, rank 4)

# Many to Many Matching

- ❖ **The temporal nature of data.**
- ❖ **Structural similarity in rules.**

$$\begin{array}{l} mammal(x) \wedge \\ feline(x) \wedge \\ carnivorous(x) \wedge \\ has-spots(x) \end{array} \quad \longrightarrow jaguar(x)$$

$$\begin{array}{l} mammal(x) \wedge \\ feline(x) \wedge \\ carnivorous(x) \wedge \\ has-stripes(x) \end{array} \quad \longrightarrow tiger(x)$$

- ❖ **Persistence of variable binding consistency.**

$$\begin{array}{l} son(x, y) \wedge \\ son(y, z) \end{array} \quad \longrightarrow grandparent(x, z)$$

# A Bit of a Dialogue with ELIZA

| | |
|---|---|
| **Person:** | Men are all alike. |
| **ELIZA:** | In what way? |
| **Person:** | They're always bugging us about something or other. |
| **ELIZA:** | Can you think of a specific example? |
| **Person:** | Well, my boyfriend made me come here. |
| **ELIZA:** | Your boyfriend made you come here. |
| **Person:** | He says I'm depressed much of the time. |
| **ELIZA:** | I am sorry to hear you are depressed. |
| **Person:** | It's true. I am unhappy. |
| **ELIZA:** | Do you think coming here will help you not to be unhappy? |
| **Person:** | I need some help, that much seems certain. |
| **ELIZA:** | What would it mean to you if you got some help? |
| **Person:** | Perhaps I could learn to get along with my mother. |
| **ELIZA:** | Tell me more about your family. |

# Some ELIZA-like rules

| | | |
|---|---|---|
| (X me Y) | $\rightarrow$ | (X you Y) |
| (I remember X) | $\rightarrow$ | (Why do remember X just now?) |
| (My {family-member} is Y) | $\rightarrow$ | (Who else in your family is Y?) |
| (X {family-member} Y) | $\rightarrow$ | (Tell me more about your family) |

# Conflict Resolution

- ❖ **Preferences based on rules.**
  - **Rule order.**
  - **Prefer special cases over more general ones.**
- ❖ **Preferences based on objects.**
  - **Prefer some objects to others.**
  - **Location in STM.**
- ❖ **Preferences based on states.**

# Syntax for a Control Rule

Under conditions A and B,
Rules that do {not} mention X
{at all,
in their left-hand side,
in their right-hand side}
will
{definitely be useless,
probably be useless
...
probably be especially useful
definitely be especially useful}