KNOWLEDGE REPRESENTATION
&
PREDICATE LOGIC

Amey D.S.Kerkar,
Asst.Professor, Computer Engineering Dept.
Don Bosco College of Engineering,
Fatorda-Goa.

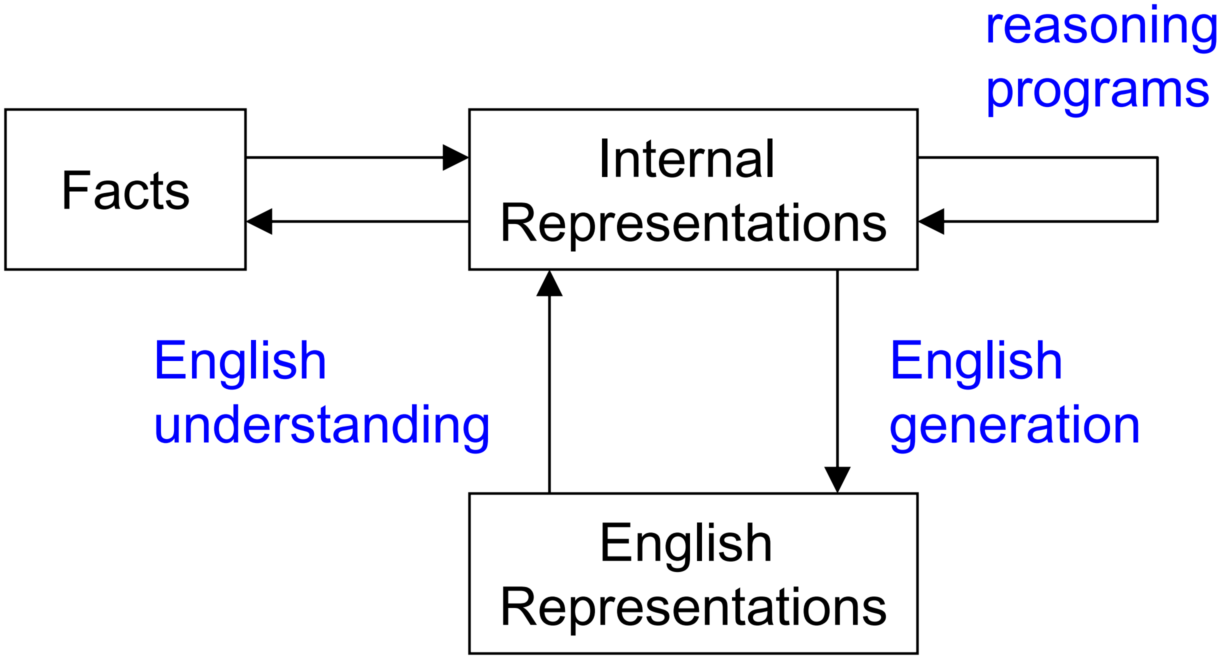
- To solve complex problems we need:
  1. Large amount of knowledge
  2. Mechanism for representation and manipulation of existing knowledge to create new solution.

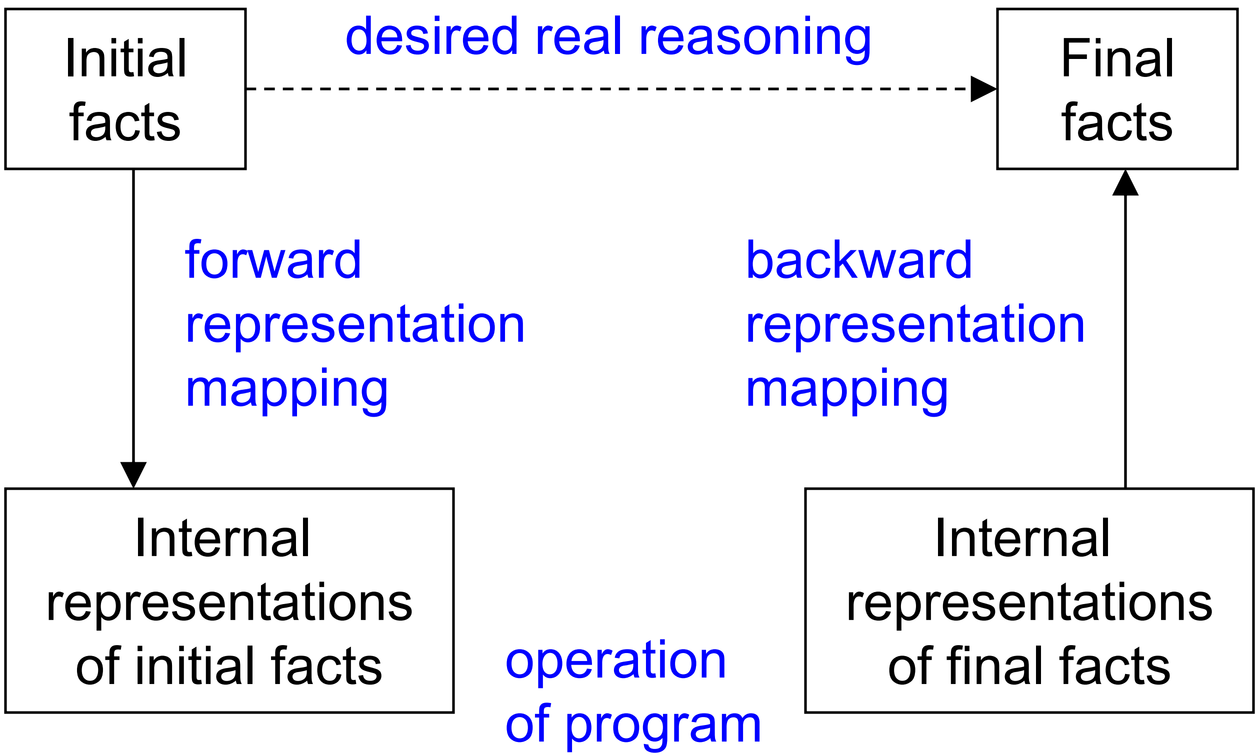
Knowledge Representation

– **Facts**: Things we want to represent. Truth in some relevant world.

– **Representation of facts.**

Representation and Mapping




Initial facts
desired real reasoning
Final facts
forward representation mapping
backward representation mapping
Internal representations of initial facts
operation of program
Internal representations of final facts

Representation and Mapping

- Spot is a dog

- Every dog has a tail

$$\Downarrow$$

Spot has a tail

Representation and Mapping

- Spot is a dog

  dog(Spot)

- Every dog has a tail

  $\forall$x: dog(x) $\rightarrow$ hastail(x)

  $\Downarrow$

  hastail(Spot)

  Spot has a tail

- Fact-representation mapping is not one-to-one.
- Good representation can make a reasoning program trivial.

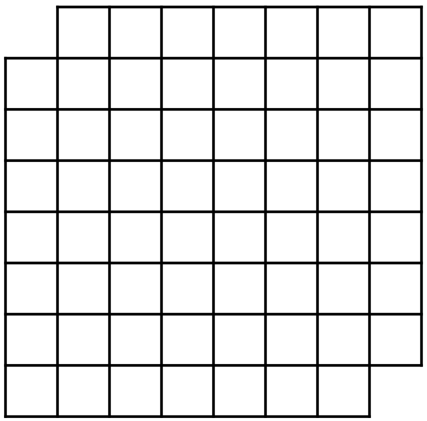
The Multilated Checkerboard Problem

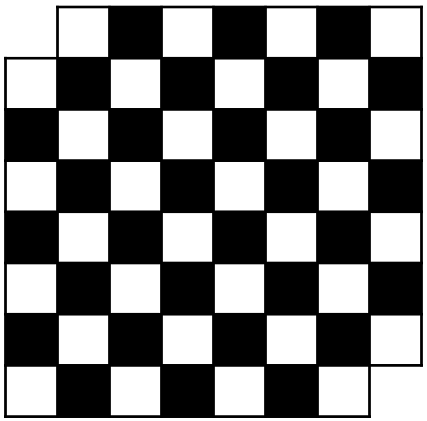
“Consider a normal checker board from which two squares, in opposite corners, have been removed.

The task is to cover all the remaining squares exactly with donimoes, each of which covers two squares. No overlapping, either of dominoes on top of each other or of dominoes over the boundary of the multilated board are allowed.

Can this task be done?”

Representation and Mapping





No. black squares = 30

No. white square = 32

Good Knowledge representation should exhibit:

1. **Representational adequacy**-

Ability to represent all kinds of knowledge that are needed in the domain.

2. **Inferential adequacy-**

Ability to manipulate representational structures such that new knowledge can be derived/inferred from the old.

3. **Inferential efficiency-**

Ability to incorporate additional information into an existing knowledge base that can be used to focus the attention of inference mechanisms in the most promising direction.

4. **Acquisitional efficiency-**

Ability to easily acquire new information.

Approaches to KR

1. Simple relational knowledge:

- Provides very weak inferential capabilities.
- May serve as the input to powerful inference engines.

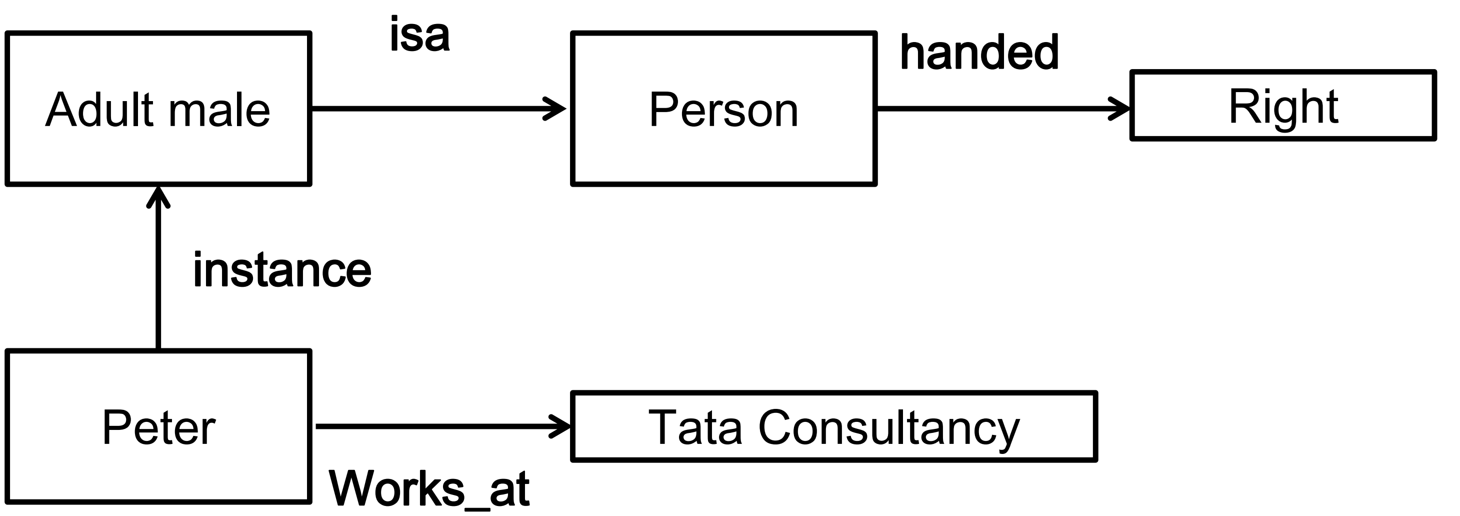
Player	**Height**	**Weight**	**handed**
Peter	6-0	180	right
Ajay	5-10	170	left
John	6-2	215	left
Vickey	6-3	205	right

Fails to infer “which right handed player can best face a particular bowler” .

Approaches to KR

Inheritable knowledge:

- Objects are organized into classes and classes are organized in a generalization hierarchy.
- Inheritance is a powerful form of inference, but not adequate.
- Ex. Property inheritance inference mechanism.



Approaches to KR

Inferential knowledge:

- Facts represented in a logical form, which facilitates reasoning.
- An inference engine is required.

ex. 1. "Marcus is a man"

2. "All men are mortal"

Implies:

3. "Marcus is mortal"

Approaches to KR

Procedural knowledge:

- Representation of “how to make it” rather than “what it is”.
- May have inferential efficiency, but no inferential adequacy and acquisitional efficiency.
- Ex. Writing LISP programs

Issues in KR

1. Important Attributes: *Isa* and *instance* attributes.

2. Relationships among attributes: *inverses*, *existence* in a *Isa* hierarchy, *single-valued attributes*, techniques for reasoning about values.

3. Choosing the Granularity: High-level facts may not be adequate for inference. Low-level primitives may require a lot of storage.

- Ex: “john spotted sue”

  [representation: *spotted(agent(john),object(sue))*]

Q1: “who spotted sue?” Ans1: “john”.

Q2: “Did john see sue?” Ans2: NO ANSWER!!!!

- Add detailed fact: *spotted(x,y)-->saw(x,y)* then Ans2: “Yes”.

- 4.Representing Set of Objects:
- 5. finding the right structure as needed.:

Ex: word “fly” can have multiple meanings:

1. “John flew to new york”
2. “John flew into a rage” [idiom]
3. “john flew a kite”

SELF: Please read frame problem pg. 96-97, Rich & Knight,3rd edition.

Propositional logic

- Statements used in mathematics.
- **Proposition** :is a declarative sentence whose value is either true or false.

Examples:

- “The sky is blue.” [Atomic Proposition]
- “The sky is blue and the plants are green.” [Molecular/Complex Proposition]
- “Today is a rainy day” [Atomic Proposition]
- “Today is Sunday” [Atomic Proposition]
- “ 2*2=4” [Atomic Proposition]

Terminologies in propositional algebra:

Statement: sentence that can be true/false.

Properties of statement:

- ✓ **Satisfyability**: a sentence is satisfyable if there is an interpretation for which it is true.

Eg.”we wear woollen cloths”

- ✓ **Contradiction:** if there is **no** interpretation for which sentence is true.

Eg. “Japan is capital of India”

- ✓ **Validity:** a sentence is valid if it is true for every interpretation.

Eg. “Delhi is the capital of India”

Inference rules:

Commutative	$p \wedge q \iff q \wedge p$	$p \vee q \iff q \vee p$
Associative	$(p \wedge q) \wedge r \iff p \wedge (q \wedge r)$	$(p \vee q) \vee r \iff p \vee (q \vee r)$
Distributive	$p \wedge (q \vee r) \iff (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \iff (p \vee q) \wedge (p \vee r)$
Identity	$p \wedge T \iff p$	$p \vee F \iff p$
Negation	$p \vee \sim p \iff T$	$p \wedge \sim p \iff F$
Double Negative	$\sim (\sim p) \iff p$	
Idempotent	$p \wedge p \iff p$	$p \vee p \iff p$
Universal Bound	$p \vee T \iff T$	$p \wedge F \iff F$
De Morgan's	$\sim (p \wedge q) \iff (\sim p) \vee (\sim q)$	$\sim (p \vee q) \iff (\sim p) \wedge (\sim q)$
Absorption	$p \vee (p \wedge q) \iff p$	$p \wedge (p \vee q) \iff p$
Conditional	$(p \implies q) \iff (\sim p \vee q)$	$\sim (p \implies q) \iff (p \wedge \sim q)$

Modus Ponens	$p \implies q$ p $\therefore q$	Modus Tollens	$p \implies q$ $\sim q$ $\therefore \sim p$
Elimination	$p \vee q$ $\sim q$ $\therefore p$	Transitivity	$p \implies q$ $q \implies r$ $\therefore p \implies r$
Generalization	$p \implies p \vee q$ $q \implies p \vee q$	Specialization	$p \wedge q \implies p$ $p \wedge q \implies q$
Conjunction	p q $\therefore p \wedge q$	Contradiction Rule	$\sim p \implies F$ $\therefore p$

INFERENCE RULES IN PROPOSITIONAL LOGIC

1. Idempotent rule:

$$P \land P ==> P$$

$$P \lor P ==> P$$

2. Commutative rule:

$$P \land Q ==> Q \land P$$

$$P \lor Q ==> Q \lor P$$

3.Associative rule:

$$P \land (Q \land R) ==> (P \land Q) \land R$$

$$P \lor (Q \lor R) ==> (P \lor Q) \lor R$$

4. Distributive Rule:

$$P \lor (Q \land R) \implies (P \lor Q) \land (P \lor R)$$

$$P \land (Q \lor R) \implies (P \land Q) \lor (P \land R)$$

5. De-Morgan's Rule:

$$\neg(P \lor Q) \implies \neg P \land \neg Q$$

$$\neg(P \land Q) \implies \neg P \lor \neg Q$$

6. Implication elimination:

$$P \rightarrow Q \Rightarrow \neg P \lor Q$$

7. Bidirectional Implication elimination:

$$(P \Leftrightarrow Q) \Longrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$$

8. Contrapositive rule:

$$P \rightarrow Q \Rightarrow \neg P \rightarrow \neg Q$$

9. Double Negation rule:

$$\neg(\neg P) \Rightarrow P$$

10. Absorption Rule:

$$\underline{P} \vee (\underline{P} \wedge Q) \Rightarrow \underline{P}$$

$$\underline{P} \wedge (\underline{P} \vee Q) \Rightarrow \underline{P}$$

11.Fundamental identities:

$P \wedge \neg p \Rightarrow F$ **[contradiction]**

$P \vee \neg P \Rightarrow T$ **[Tautology]**

$P \vee T \Rightarrow P$

$P \vee F \Rightarrow P$

$P \vee \neg T \Rightarrow P$

$P \wedge F \Rightarrow F$

$P \wedge T \Rightarrow P$

12. Modus Ponens:

If **P** is true and **P→Q** then we can infer **Q** is also true.

$$\frac{\begin{array}{c} P \\ P \rightarrow Q \end{array}}{\text{Hence, } Q}$$

13. Modus Tollens:

If **┑P** is true and **P→Q** then we can infer **┑Q** .

$$\frac{\begin{array}{c} \neg P \\ P \rightarrow Q \end{array}}{\text{Hence, } \neg Q}$$

14. Chain rule:

If $p \rightarrow q$ and $q \rightarrow r$ then $p \rightarrow r$

15. Disjunctive Syllogism:

if $\neg \mathbf{p}$ and $\mathbf{p} \vee \mathbf{q}$ we can infer $\mathbf{q}$ is true.

16. AND elimination:

Given P and Q are true then we can deduce P and Q seperately:

$$P \wedge Q \rightarrow P$$

$$P \wedge Q \rightarrow Q$$

17. AND introduction:

Given **P** and **Q** are true then we deduce **P ∧ Q**

18. OR introduction:

Given P and Q are true then we can deduce P and Q separately:

$$P \rightarrow P \vee Q$$

$$Q \rightarrow P \vee Q$$

- Example:

“I will get wet if it rains and I go out of the house”

Let Propositions be:

W : “I will get wet “

R : “it rains “

S : “I go out of the house”

$$(S \wedge R) \rightarrow W$$

Using Propositional Logic

Representing simple facts

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

Normal Forms in propositional Logic

1. Conjunctive normal form (CNF):

e.g. $(P \vee Q \vee R) \wedge (P \vee Q) \wedge (P \vee R) \wedge P$

It is conjunction ($\wedge$) of disjunctions ($\vee$)

Where disjunctions are:

$$\left.\begin{array}{l} 1.\ (P \vee Q \vee R) \\ 2.\ (P \vee Q) \\ 3.\ (P \vee R) \\ 4.\ P \end{array}\right\} \text{clauses}$$

2. Disjunctive normal form (DNF):

e.g. $(P \wedge Q \wedge R) \vee (P \wedge Q) \vee (P \wedge R) \vee P$

It is disjunction ($\vee$) of conjunctions ($\wedge$)

Procedure to convert a statement to CNF

1. **Eliminate implications and biconditionals using formulas:**

- **$(P \Leftrightarrow Q) \Longrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$**
- **$P \rightarrow Q \Rightarrow \neg P \vee Q$**

2. Apply De-Morgan's Law and reduce NOT symbols so as to bring negations before the atoms. Use:

- **$\neg(P \vee Q) \Longrightarrow \neg P \wedge \neg Q$**
- **$\neg (P \wedge Q) \Longrightarrow \neg P \vee \neg Q$**

3. Use distributive and other laws & equivalent formulas to obtain Normal forms.

Conversion to CNF example

Q. Convert into CNF : $((P \rightarrow Q) \rightarrow R)$

Solution:

Step 1: $((P \rightarrow Q) \rightarrow R)$ ==> $((\neg P \vee Q) \rightarrow R)$

==> $\neg(\neg P \vee Q) \vee R$

Step 2: $\neg(\neg P \vee Q) \vee R$ ==> $(P \wedge \neg Q) \vee R$

Step 3: $(P \wedge \neg Q) \vee R$ ==> $\underbrace{(P \vee R) \wedge (\neg Q \vee R)}_{\text{CNF}}$

Resolution in propositional logic

Proof by Refutation / contradiction.

- Used for theorem proving / rule of inference.

- **Method:** Say we have to prove proposition A
- Assume A to be false i.e. ┐A
- Continue solving the algorithm starting from ┐A
- If you get a contradiction (F) at the end it means your initial assumption i.e. ┐A is false and hence proposition **A must be true.**

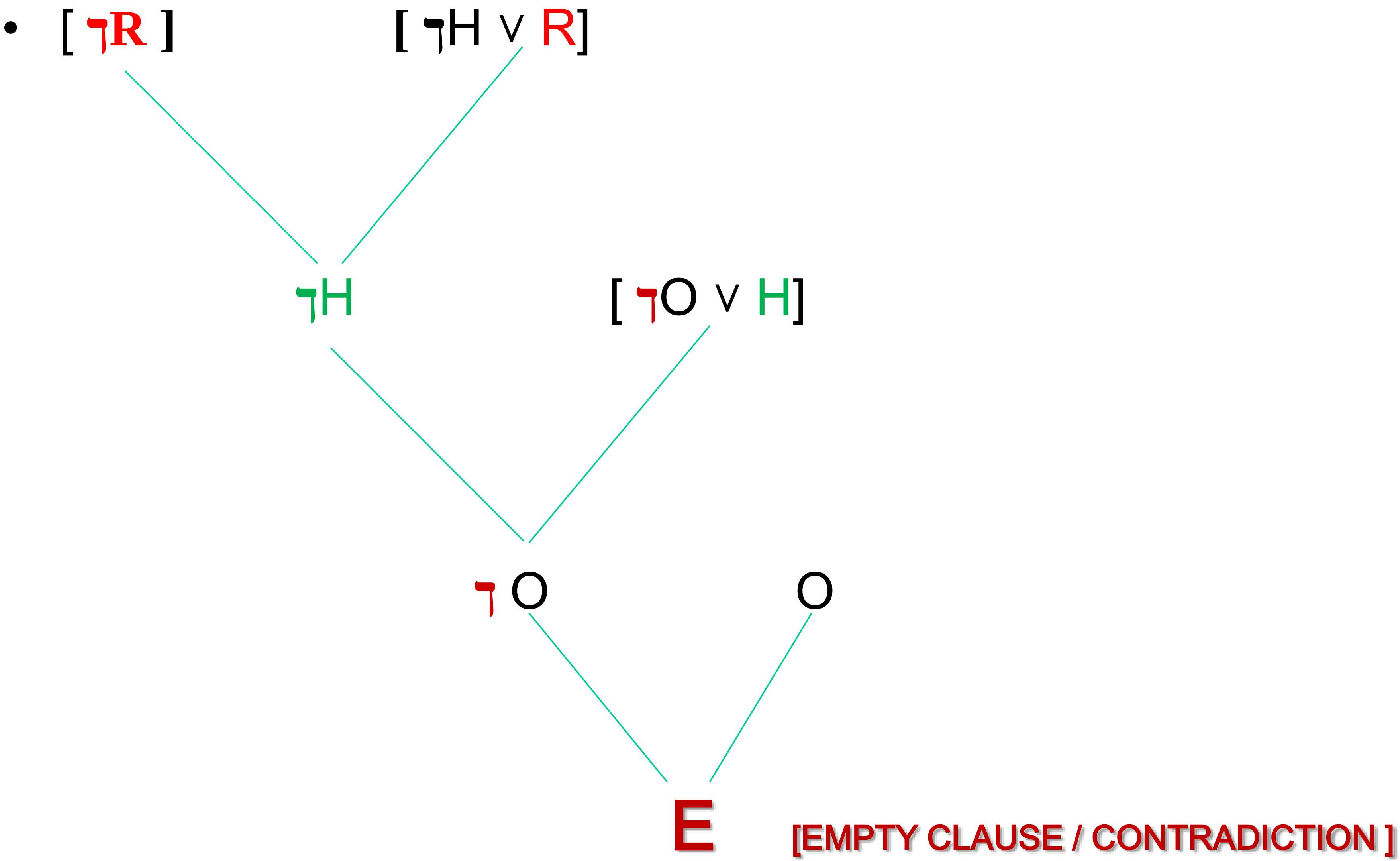
- **Clause**: disjunction of literals is called clause.

- How it works?
- E.g. " If it is Hot then it is Humid. If it is humid then it will rain. It is hot." prove that " it will rain."

- Solution:
- Let us denote these statements with propositions H,O and R:
  - H: " It is humid".
  - O: " It is Hot". And R: "It will rain".

- Formulas corresponding to the sentences are:
- 1. "if it is hot then it is humid" $[O \rightarrow H] \implies \neg O \vee H$
- 2. "If it is humid then it will rain". $[H \rightarrow R] \implies \neg H \vee R$
- 3. " It is Hot" $[O] \implies O$
-
- To prove: R.

- Let us assume “it will NOT rain” [¬R]



- Since an empty clause (E) has been deduced we say that our assumption is wrong and hence we have proved:

"It will rain"

Using Prepositional Logic:

- Theorem proving is decidable BUT
- It Cannot represent objects and quantification.
- Hence we go for PREDICATE LOGIC

PREDICATE LOGIC

- Can represent objects and quantification
- Theorem proving is semi-decidable

Representing simple facts (Preposition)

"SOCRATES IS A MAN"

SOCRATESMAN ---------1

"PLATO IS A MAN"

PLATOMAN ---------2

Fails to capture relationship between Socrates and man.
We do not get any information about the objects involved
Ex:

if asked a question : "who is a man?" we cannot get answer.

Using Predicate Logic however we can represent above facts as: Man(Socretes) and Man(Plato)

Using Predicate Logic

1. Marcus was a man.

   man(Marcus)

Using Predicate Logic

2. Marcus was a Pompeian.

   Pompeian(Marcus)

- Quantifiers:
- 2 types:-

- Universal quantifier **(∀)**

- ∀x: means “for all” x
- It is used to represent phrase “ for all”.
- It says that something is true for all possible values of a variable.

- Ex. “ John loves everyone”
-

- Quantifiers:
- 2 types:-

- Universal quantifier **(∀)**

- $\forall x$: means "for all" x
- It is used to represent phrase " for all".
- It says that something is true for all possible values of a variable.

- Ex. " John loves everyone"

  $\forall x$: **loves(John , x)**

- Existential quantifier (∃):

- Used to represent the fact " there exists some"
- Ex:
- "some people like reading and hence they gain good knowledge"

$\exists x: \{ [person(x) \wedge like(x, reading)] \rightarrow gain(x, knowledge) \}$

- **"lord Haggins has a crown on his head"**
- $\exists x: crown(x) \wedge onhead(x, Haggins)$

Nested Quantifiers

- We can use both $\forall$ **and** $\exists$ seperately

- Ex: " everybody loves somebody "

$$\forall x: \exists y: \text{ loves } (x , y)$$

- <u>**Connection between $\forall$ and $\exists$**</u>
- " everyone dislikes garlic"

$$\forall\ x: \neg \text{ like } (x , \text{garlic})$$

➢ **This can be also said as:**

"there does not exists someone who likes garlic"

$$\neg \exists x: \text{like } (x, \text{garlic})$$

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

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

4. Every one is loyal to someone.

$\forall x$: $\exists y$: loyalto(x, y) $\exists y$: $\forall x$: loyalto(x, y)

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

$\forall x$: $\forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y)

$\rightarrow \neg$loyalto(x, y)

6. “All Pompeians were Romans”

$\forall x$: Pompeian(x) $\rightarrow$ Roman(x)

8. Marcus tried to assassinate Caesar.

   tryassassinate(Marcus, Caesar)

Some more examples

- "all indoor games are easy"

$$\forall x: indoor\_game(x) \rightarrow easy(x)$$

- "Rajiv likes only cricket"

Like(Rajiv, Cricket)

- "Any person who is respected by every person is a king"

$$\exists x: \forall y: \{ person(x) \wedge person(y) \wedge respects(y, x) \rightarrow king(x) \}$$

- "god helps those who helps themselves"

$$\forall x: helps(god, helps(x, x))$$

- "everyone who loves all animals is loved by someone"

$$\underbrace{\forall x: [\ \forall y: animal(y) \rightarrow loves(x, y)\ ]}_{\text{everyone who loves all animals}}$$

$$\exists z: loves(z, x)$$ there exist someone z and z loves x

Thus the predicate sentence is:

$$\forall x: [\ [\forall y: animal(y) \rightarrow loves(x, y)\ ] \rightarrow [\ \exists z: loves(z, x)\ ]\ ]$$

Computable functions and predicates

- " Marcus was born in 40 A.D"

  Born(Marcus, 40)

- " All Pompeians died when volcano erupted in 79 A.D"

  $Erupted(volcano, 79) \wedge \forall x: [\, Pompeian(x) \rightarrow Died(x, 79)]$

- **" no mortal lives longer than 150 years"**
- **How to solve ?**
- **let t1 is time instance 1 and t2 is time instance 2**
- **We use computable function gt(… , ….) which computes greater than.**

$$\forall x: \forall t1: \forall t2: \; mortal(x) \wedge born(x, t1) \wedge gt(t2 - t1, 150) \rightarrow dead(x, t2)$$

Resolution algorithm in predicate logic

- Proof by refutation.
- INPUT: Predicate sentences in clausal form (CNF)
- (See conversion algo on next slide)

- Algorithm steps :-

Convert all the propositions of KB to clause form (S).

2. Negate α and convert it to clause form. Add it to S.

3. Repeat until either a contradiction is found or no progress can be made.

   a. Select two clauses ($\alpha \vee \neg P$) and ($\gamma \vee P$).

   b. Add the resolvent ($\alpha \vee \gamma$) to S.

Conversion to Clause Form

1. Eliminate $\rightarrow$.

   $P \rightarrow Q \equiv \neg P \vee Q$

2. Reduce the scope of each $\neg$ to a single term.

   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
   $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

   $\neg\forall x\text{: } P \equiv \exists \mathbf{x}\text{: } \neg P$
   $\neg\exists \mathbf{x}\text{: } p \equiv \forall \mathbf{x}\text{: } \neg P$

   $\neg\neg\, P \equiv P$

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

   $(\forall \mathbf{x}\text{: } \mathbf{P}(\mathbf{x})) \vee (\exists \mathbf{x}\text{: } \mathbf{Q}(\mathbf{x})) \equiv$

   $(\forall \mathbf{x}\text{: } \mathbf{P}(\mathbf{x})) \vee (\exists \mathbf{y}\text{: } \mathbf{Q}(\mathbf{y}))$

4. Move all quantifiers to the left without changing their relative order.

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

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

5. Eliminate $\exists$ (Skolemization).

$\exists x: P(x) \equiv P(c)$ Skolem constant

$\forall x: \exists y\ P(x, y) \equiv \forall x: P(x, f(x))$ Skolem function

6. Drop $\forall$.

$$\forall x: P(x) \equiv P(x)$$

7. Convert the formula into a conjunction of disjuncts.

$$(\mathbf{P} \wedge \mathbf{Q}) \vee \mathbf{R} \equiv (\mathbf{P} \vee \mathbf{R}) \wedge (\mathbf{Q} \vee \mathbf{R})$$

8. Create a separate clause corresponding to each conjunct.

9. Standardize apart the variables in the set of obtained clauses.

- Example of conversion:

$\forall x: \neg [\, Roman(x) \rightarrow (\, Pompeian(x) \wedge \neg\, hate(x, Caesar))]$

After step 1: i.e. elimination of $\rightarrow$ and $\Leftrightarrow$ the above stmt becomes:

$\forall x: \neg [\, \neg\, Roman(x) \vee (Pompeian(x) \wedge \neg\, hate(x, Caesar))]$

After step 2: i.e. reducing scope of $\neg$ the above stmt becomes:

$\forall x: [\, Roman(x) \wedge \neg(Pompeian(x) \wedge \neg\, hate(x, Caesar))\,]$

$\Downarrow$

$\forall x: [\, Roman(x) \wedge (\neg Pompeian(x) \vee hate(x, Caesar))\,]$

- Example to demonstrate step 3:- i.e. standardization of variables.

$\forall x$: [[$\forall y$: animal (y) → loves(x , y)] → [$\exists y$: loves(y , x)]]

After step 3 above stmt becomes,

$\forall x$: [[$\forall y$: animal (y) → loves(x , y)] → [$\exists z$: loves(z, x)]]

- Example to demostrate step 4: Move all quantifiers to the left without changing their relative order.
-
- $\forall x$: [[$\forall y$: animal (y) $\wedge$ loves(x , y)] $\vee$ [$\exists z$: loves(z, x)]]
- After applying step 4 above stmt becomes:
- $\forall x$: $\forall y$: $\exists z$: [animal (y) $\wedge$ loves(x , y) $\vee$ loves(z, x)]
- **After first 4 processing steps of conversion are carried out on original statement S, the statement is said to be in PRENEX NORMAL FORM**

- Example to demostrate step 5: skolemization (i.e. elimination of ∃ quantifier)

$\exists$y: President (y)

Can be transformed into

President (**S1**)

where S1 is a function that somehow produces a value that satisfies President (S1) – S1 called as **Skolem constant**

- Ex. 2:

$\exists$y: $\forall$x: leads (**y** , x)

Here value of y that satisfies 'leads' depends on particular value of x hence above stmt can be written as:

$\forall$x: leads (**f(x)** , x)

Where f(x) is **skolem function**.

- Example to demonstrate step 6: dropping prefix ∀

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

- After prefix dropped becomes,

$$[\neg \text{Roman}(x) \vee \neg \text{know}(x, y) \vee \text{hate}(y, z)]$$

- Example to demostrate step 7: Convert the formula into a conjunction of disjuncts.(CNF)

- Roman (x) ∨ ((hate (x , caesar) ∧ ¬loyalto (x , caesar))

- $\underbrace{\text{Roman (x)}}_{P} \vee (\underbrace{\text{(hate (x , caesar)}}_{Q} \wedge \underbrace{\neg\text{loyalto (x , caesar)}}_{R})$

- $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

CLAUSE 1 (Roman (x) ∨ (hate (x , caesar)) ∧

CLAUSE 2 (Roman (x) ∨ ¬loyalto (x , caesar))

Unification

- It’s a matching procedure that compares two literals and discovers whether there exists a set of substitutions that can make them identical.

- E.g.

Hate(marcus , X)   Hate (marcus , caesar)

caesar/ X

e.g. 2.

Hate(X,Y)   Hate(john, Z)   could be unified as:

John/X   and   y/z

Unification:

UNIFY(p, q) = unifier θ where SUBST(θ, p) = SUBST(θ, q)

$\forall$x: knows(John, x) $\rightarrow$ hates(John, x)

knows(John, Jane)

$\forall$y: knows(y, Leonid)

$\forall$y: knows(y, mother(y))

$\forall$x: knows(x, Elizabeth)

UNIFY(knows(John ,x) ,knows(John, Jane)) = {Jane/x}

UNIFY(knows(John, x), knows(y, Leonid)) = {Leonid/x, John/y}

UNIFY(knows(John, x), knows(y, **mother(y)**)) = {John/y, **mother(John)**/x}

UNIFY(knows(John, **x**), knows(**x**, Elizabeth)) = **FAIL**

Resolution algorithm

- It is used as inference mechanism.
- <u>Pre-processing steps:</u>

1. Convert the given English sentence into predicate sentence.
2. **<u>Not all</u>** of these sentences will be in clausal form (CNF).

   If any sentence is not in clausal form then <u>convert it into clausal form</u>.
3. Give these sentences (clauses) as an **input** to resolution algorithm.

<u>Resolution algorithm steps:</u>

A. Negate the proposition which is to be proved.

   i.e. If we have to prove :-

   like(tommy , cookies) then assume ¬ like(tommy,cookies)

   Add the resultant sentence to the set of sentences from step 3

B. Repeat until contradiction is found or no progress can be made:

i. Select two clauses , call them parent clauses and resolve them together.

The resultant clause is called resolvant.

e.g. $P(x) \vee Q(x)$ $R(x) \vee \neg P(X)$

$Q(x) \vee R(x)$

ii. If resolvant contains empty clause then contradiction has been found.

$G(x)$ $\neg G(x)$

E **[EMPTY CLAUSE]**

iii. If step ii. Results in empty clause , it means our assumption is wrong

and the original clause (to be proved) has to be true.

Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

1. “Marcus was a man”

$\boxed{\text{man(marcus)}}$ ------------- 1

2. “Marcus was a Pompeian”

$\boxed{\text{pompeian (marcus)}}$ ------------- 2

3. “All Pompeian's were Romans”

=> $\forall x1$: pompeian(x1) $\rightarrow$ roman(x1).

=> $\forall x1$: $\neg$ **pompeian(x1)** $\vee$ **roman(x1)**

$\boxed{\neg \text{ pompeian (x1)} \vee \text{roman(x1)}}$ ----------------- 3

4. “Caesar was a ruler”

$\boxed{\text{ruler (caesar)}}$ ---------------- 4

5. “all romans were either loyalto caesar or hated him”

=> $\forall$x2: roman(x2) $\rightarrow$ [loyalto(x2 , caesar) $\vee$ hate(x2 , caesar)]

=> $\forall$x2: $\neg$ roman(x2) $\vee$ loyalto(x2 , caesar) $\vee$ hate(x2 , caesar)

=> $\neg$ roman(x2) $\vee$ loyalto(x2 , caesar) $\vee$ hate(x2 , caesar)

$\boxed{\neg \text{ roman (x2)} \vee \text{loyalto (x2 , caesar)} \vee \text{hate (x2 , caesar)}}$ ------ 5

- "Every one is loyal to someone"

=> $\forall x3: \exists y1: loyalto(x3, y1).$

Let f(x3) be a skolem function then,

=> $\forall x3: loyalto(x3, f(x3)).$

=> $loyalto(x3, f(x3))$

$\boxed{loyalto\ (x3, f(x3))}$ ---------------- 6

7. “People only try to assassinate rulers they are not loyal to.”

=> $\forall x4: \forall y2: [man(x4) \wedge ruler(y2) \wedge tryassassinate(x4, y2)]$

$\rightarrow \neg loyalto(x4, y2)$

=> $\forall x4: \forall y2: \neg [man(x4) \wedge ruler(y2) \wedge tryassassinate(x4, y2)]$

$\vee \neg loyalto(x4, y2)$

$\Rightarrow \forall x4: \forall y2: \neg man(x4) \vee \neg ruler(y2) \vee \neg tryassassinate(x4, y2) \vee$
$\neg loyalto(x4, y2)$

let f(x4) be skolem function then,

$\Rightarrow$ => $\forall x4: \neg man(x4) \vee \neg ruler(f(x4)) \vee$

$\neg tryassassinate(x4, f(x4)) \vee \neg loyalto(x4, f(x4))$

$\Rightarrow \neg man(x4) \lor \neg ruler(f(x4)) \lor \neg tryassassinate(x4, f(x4)) \lor$
$\neg loyalto(x4, f(x4))$

$\neg$ man(x4) $\lor \neg$ ruler(f(x4)) $\lor$ $\neg$ tryassassinate(x4, f(x4)) $\lor$ $\neg$loyalto(x4, f(x4))

---------- 7

8. "Marcus tried to assassinate Caesar"

tryassassinate(marcus , caesar)

tryassassinate(marcus , caesar)

------------ 8

To prove : marcus hate caesar i.e. hate(marcus, caesar)

- Assume ¬ hate(marcus, caesar)

(5)

¬ hate (marcus , caesar)

¬ roman (x2) ∨ loyalto (x2 , caesar) ∨ hate (x2 , caesar)

x2 / marcus

¬ roman (marcus) ∨ loyalto (marcus, caesar)

¬ pompeian (x1) ∨ roman(x1)

(3)

x1 / marcus

(2)

pompeian (marcus)

¬ pompeian (marcus) ∨ loyalto (marcus, caesar)

loyalto (marcus, caesar)

loyalto (marcus, caesar)

(7)

¬ man(x4) ∨ ¬ ruler(f(x4)) ∨ ¬ tryassassinate(x4, f(x4)) ∨ ¬loyalto(x4, f(x4))

x4/ marcus

f(x4)/ caesar

¬ man(marcus) ∨ ¬ ruler(caesar) ∨ ¬ tryassassinate(marcus , caesar)

(8)

tryassassinate(marcus , caesar)

¬ man(marcus) ∨ ¬ ruler(caesar)

(1)

man(marcus)

¬ ruler(caesar)

¬ ruler(caesar)

(4)

ruler(caesar)

E

- Since we get an empty clause i.e. contradiction our assumption that ¬ hate(marcus, caesar) is false

hence

hate(marcus, caesar) must be true.

- Consider the following paragraph:

“ anything anyone eats is called food. Milka likes all kind of food. Bread is a food. Mango is a food. Alka eats pizza. Alka eats everything milka eats.”

Translate the following sentences into (WFF) in predicate logic and then into set of clauses. Using resolution principle answer the following:

1. Does Milka like pizza?
2. what food Alka eats? [Question answering]

- Solution:

1. " anything anyone eats is called food."

$$\forall x: \forall y: eats(x , y) \rightarrow food(y)$$

$$\Rightarrow \forall x: \forall y: \neg\ eats(x , y) \vee food(y)$$

$$\Rightarrow \neg\ \mathbf{eats(x , y) \vee food(y)} \qquad \mathbf{(1)}$$

2. "Milka likes all kind of food"

$$\forall y1: food(y1) \rightarrow like(milka , y1)$$

$$\Rightarrow \forall y1: \neg\ food(y1) \vee like(milka , y1)$$

$$\Rightarrow \neg\ \mathbf{food(y1) \vee like(milka , y1)} \qquad \mathbf{(2)}$$

3. "Bread is a food"

food(bread) **(3)**

4. "Mango is a food"

food(mango) **(4)**

5. "Alka eats Pizza"

$eats(alka, pizza)$ **(5)**

6. "Alka eats everything Milka eats"

$\forall x1: eats(milka , x1) \rightarrow eats(alka, x1)$

=> $\forall x1: \neg\, eats(milka , x1) \vee eats(alka, x1)$

=> **$\neg\, eats(milka , x1) \vee eats(alka, x1)$** **(6)**

Question to be answered : 1. "Does Milka likes Pizza ?"

assume : "Milka **<u>does not</u>** like Pizza"

$\neg\, like(milka , pizza)$ **(7)**

¬ like(milka , pizza)

(2)

¬ food(y1) ∨ like(milka , y1)

pizza/ y1

¬ food(pizza)

(1)

eats(x , y) ∨ food(y)

pizza/ y

¬ eats(x , pizza)

(5)

eats(alka, pizza)

alka/ x

E

Since <u>¬ like(milka , pizza) is contradiction</u> like(milka , pizza) is true

Question to be answered : 2. “ what food Alka eats ?”

eats(alka, ??)

there exist something which Alka eats we have to find the value of x

$\exists x$: eats (alka, x)

Assume : alka does not eat anything

$\neg$ [$\exists x2$: eats (alka, x2)]

=> $\forall x2$: $\neg$ eats (alka , x2)

=> $\neg$ eats (alka , x2) (7)

(7) (5)

$\neg$ eats (alka , x2) eats(alka, pizza)

pizza/ x2

E

- Therefore alka does not eat anything is false and
- Alka eats something is true.
- And x2 stores pizza

- Therefore we conclude :

eats (alka, ??) answer is "pizza"

Instance and Isa relationship

- “ Marcus is a man”

  man(marcus)

  OR

  instance(marcus , man)   where marcus is an object/ instance of class ‘man’

“ all pompeians were romans”

$$\forall x: pompeian(x) \rightarrow roman(x).$$

OR

$$\forall x: instance(x, pompeian) \rightarrow instance(x, roman).$$

- **Isa Predicate :**

“ all pompeians were romans”

$$\forall x: pompeian(x) \rightarrow roman(x).$$

OR

$$\forall x: instance(x, pompeian) \rightarrow instance(x, roman). \text{------}(1)$$

- Now using isa predicate (1) becomes,

$$Isa(\ pompeian\ ,\ roman)$$

which means pompeian is a subclass of roman class

but it also requires extra axiom :

$$\forall x: \forall y: \forall z: isa(\ y, z) \wedge instance\ (x\ ,\ y) \rightarrow instance\ (\ x\ ,\ z)$$

Using Predicate Logic

- Many English sentences are ambiguous.
- There is often a choice of how to represent knowledge.
- Obvious information may be necessary for reasoning
- We may not know in advance which statements to deduce (P or $\neg$P).

THANK YOU

☺