Artificial Intelligence

Open Elective

Module2: Knowledge Representation: CH4

Dr. Santhi Natarajan
Associate Professor
Dept of AI and ML
BMSIT, Bangalore

Knowledge: What to represent

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations.

Knowledge-Base: The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge about what

- **Object**: All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.

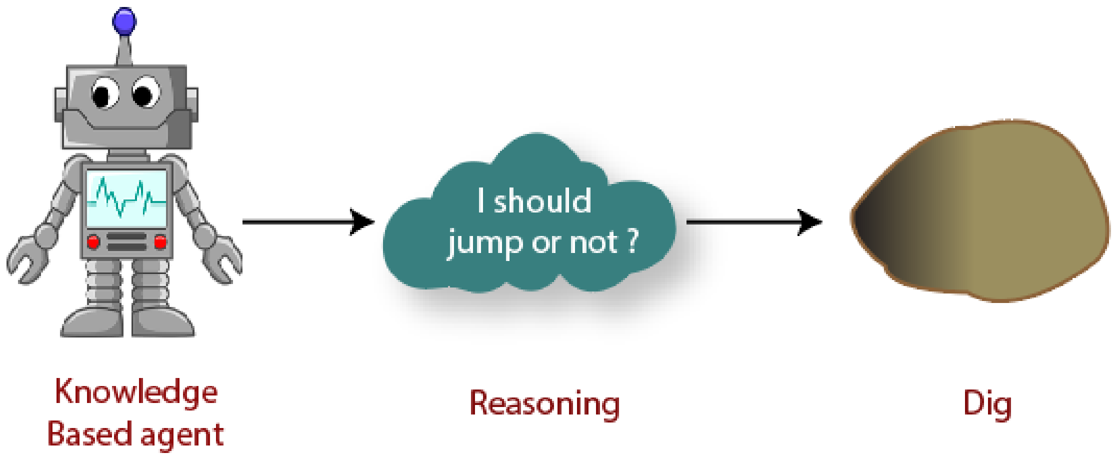
- **Events**: Events are the actions which occur in our world.

- **Performance**: It describe behavior which involves knowledge about how to do things.

- **Meta-knowledge**: It is knowledge about what we know.

- **Facts**: Facts are the truths about the real world and what we represent.

Ontological Engineering



Knowledge: The information related to the environment is stored in the machine.

Reasoning: The ability of the machine to understand the stored knowledge.

Intelligence: The ability of the machine to make decisions on the basis of the stored information.

Ontological Engineering: Systems that represent large and modular knowledge on complex domains. General concepts such as actions, time, physical objects, performance, meta data and beliefs could be expressed on a larger scale.

Knowledge Representation and Mapping

Facts: truths in some relevant world. These are the things we want to represent.

Knowledge: typically large amount of knowledge is required to solve complex problems in AI

Manipulation of knowledge: knowledge needs to be manipulated to find solutions.

Representation: facts are typically represented in some formalism. These representations are the things that we actually be able to manipulate. A good representation sometimes makes the operation of a reasoning program not only correct, but trivial as well.

Structuring at two levels:

- **Level 1: Knowledge level**: here, the facts (including the agent's behaviours and current goals) are described.

- **Level 2: Symbol level**: here, representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

Knowledge: Types

Declarative Knowledge

Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

Procedural Knowledge

Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

Meta Knowledge

Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

Heuristic Knowledge

Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a field or subject.

- Heuristic knowledge is a set of rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

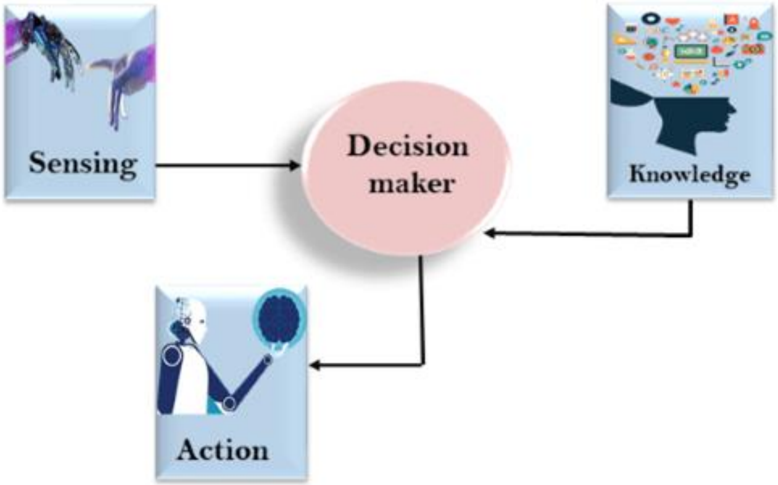
Structural Knowledge

Structural knowledge:

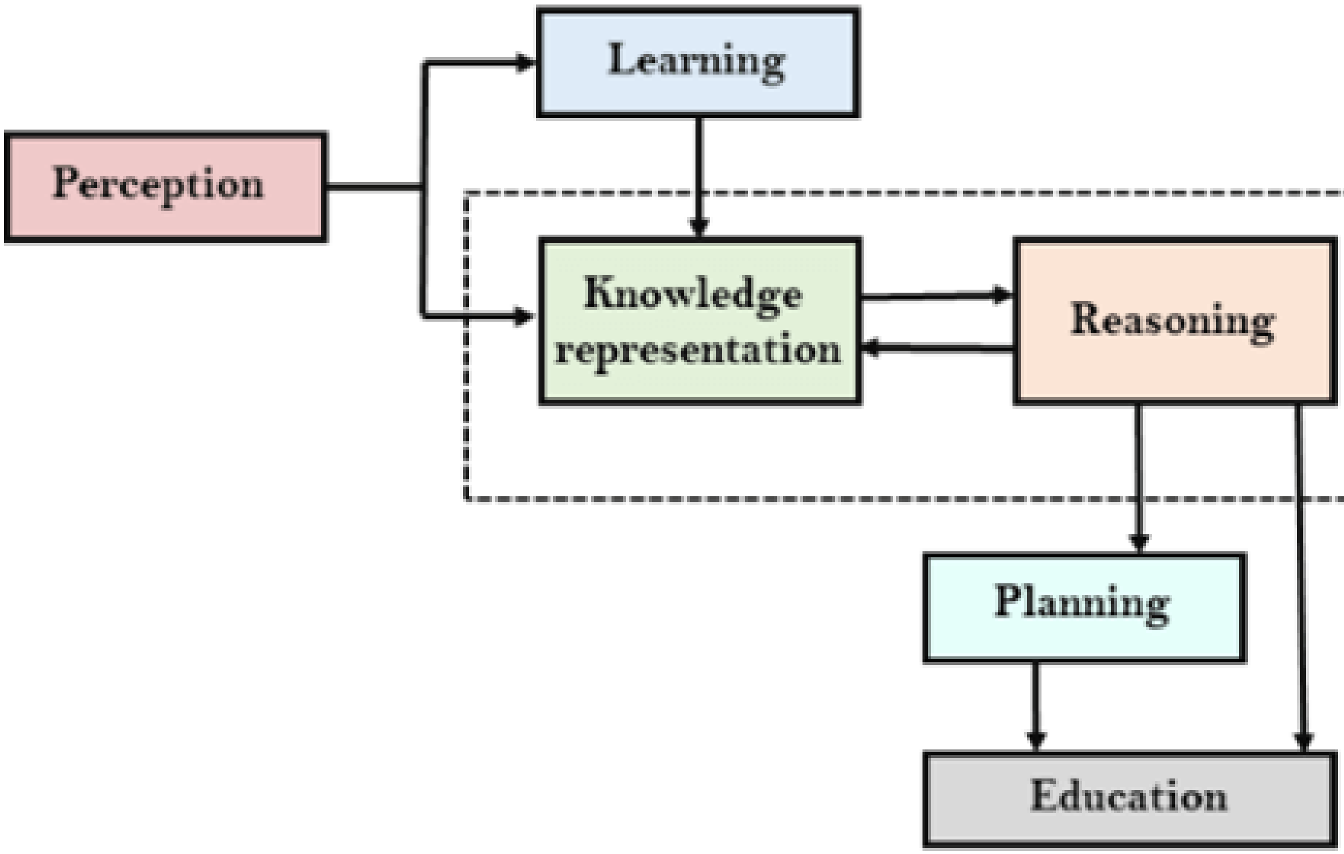
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

Knowledge versus Intelligence

- Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence.
- Knowledge plays an important role in demonstrating intelligent behavior in AI agents.
- An agent is only able to accurately act on some input when he has some knowledge or experience about that input.
- As we can see in the diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI Knowledge Cycle



AI Approaches Knowledge Representation (KR)

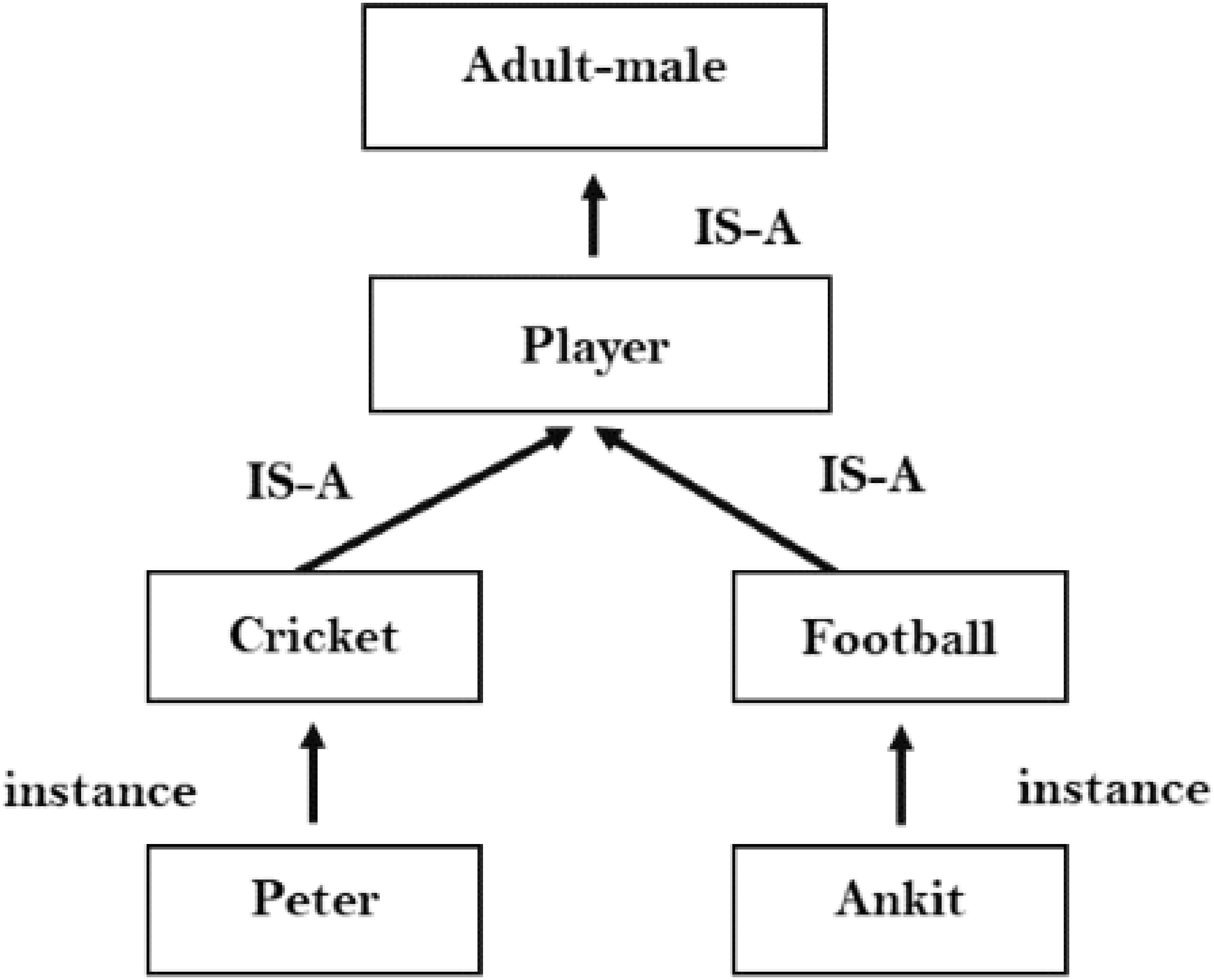
- **Simple Relational Knowledge**
- **Inheritable Knowledge**
- **Inferential Knowledge**
- **Procedural Knowledge**

Simple Relational Knowledge

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

Inheritable Knowledge



Inheritable Knowledge

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.

- All classes should be arranged in a generalized form or a hierarchal manner.

- In this approach, we apply inheritance property.

- Elements inherit values from other members of a class.

- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.

- Every individual frame can represent the collection of attributes and its value.

- In this approach, objects and values are represented in Boxed nodes.

- We use Arrows which point from objects to their values.

Inferential Knowledge

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.

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.

   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)

Inferential Knowledge

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

inclusive-or

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

exclusive-or

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

6. Every one is loyal to someone.

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

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

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

8. Marcus tried to assassinate Caesar.

$tryassassinate(Marcus, Caesar)$

Was Marcus loyal to Caesar?

$man(Marcus)$

$ruler(Caesar)$

$tryassassinate(Marcus, Caesar)$

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

$\neg loyalto(Marcus, Caesar)$

Procedural Knowledge

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is If-Then rule.
- In this knowledge, we can use various coding languages such as LISP language and Prolog language.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements of a KR System

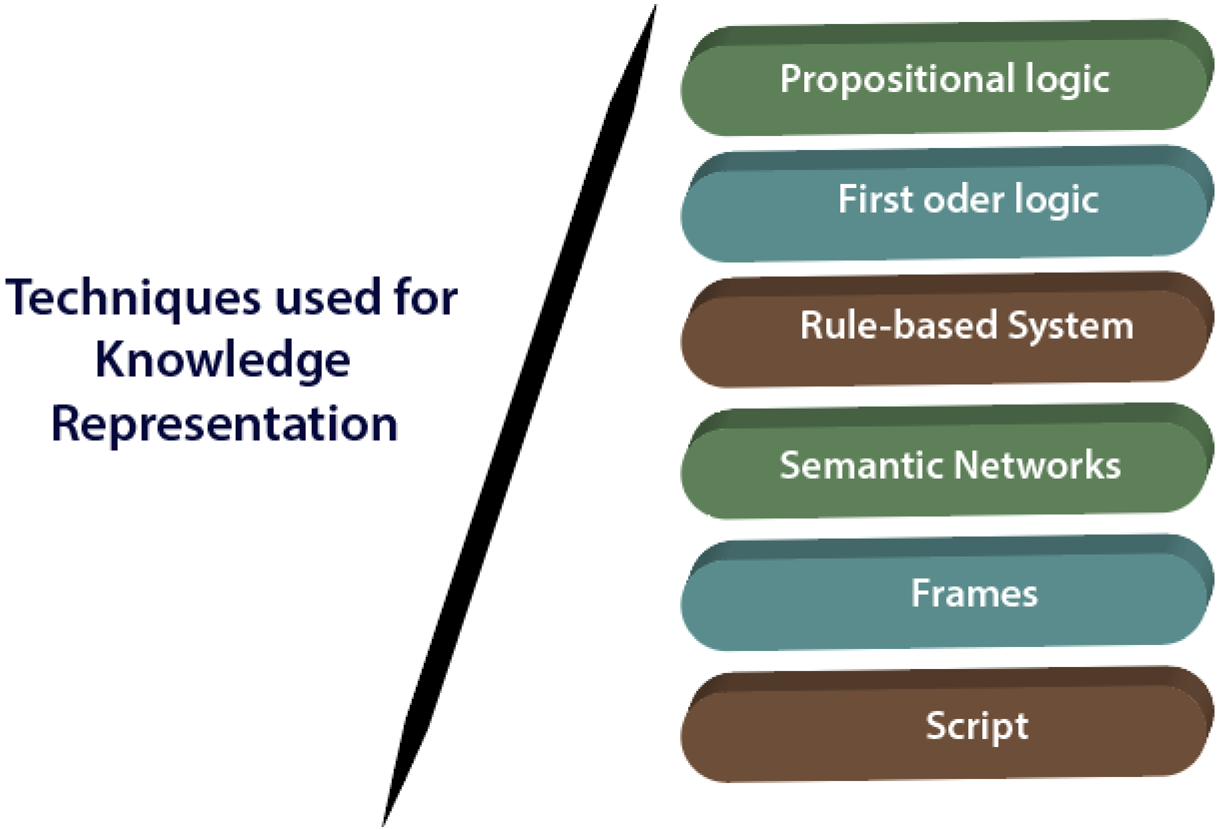
- **Representational Adequacy**: It is the ability of the system to represent all kinds of knowledge needed in a specific domain.

- **Inferential Adequacy**: It is the ability of a knowledge representation system to manipulate the current stored knowledge so that newly gained knowledge could be added.

- **Inferential Efficiency**: It is the ability of the system to directly add new knowledge in the system with efficiency

- **Acquisitional Efficiency**: It is the ability of the system to automatically acquire new knowledge from the environment. This leads the system to give more productive result as more knowledge adds up with the current knowledge.

Techniques used for KR



Techniques used for KR

Logic:

It is the basic method used to represent the knowledge of a machine. The term logic means to apply intelligence over the stored knowledge. Logic can be further divided as:

Propositional Logic:

This technique is also known as propositional calculus, statement logic, or sentential logic. It is used for representing the knowledge about what is true and what is false.

First-order Logic:

It is also known as Predicate logic or First-order predicate calculus (FOPL). This technique is used to represent the objects in the form of predicates or quantifiers. It is different from Propositional logic as it removes the complexity of the sentence represented by it. In short, FOPL is an advance version of propositional logic.

Techniques used for KR

Rule-based System:

- This is the most commonly used technique in artificial intelligence.
- In the rule-based system, we impose rules over the propositional logic and first-order logic techniques.
- If-then clause is used for this technique.
- **For example,** if there are two variables A and B. Value of both A and B is True. Consequently, the result of both should also be True and vice-versa.
- **It is represented as:**

If the value of A and B is True, then the result will be True.

- So, such a technique makes the propositional as well as FOPL logics bounded in the rules.

Techniques used for KR

Semantic Networks:

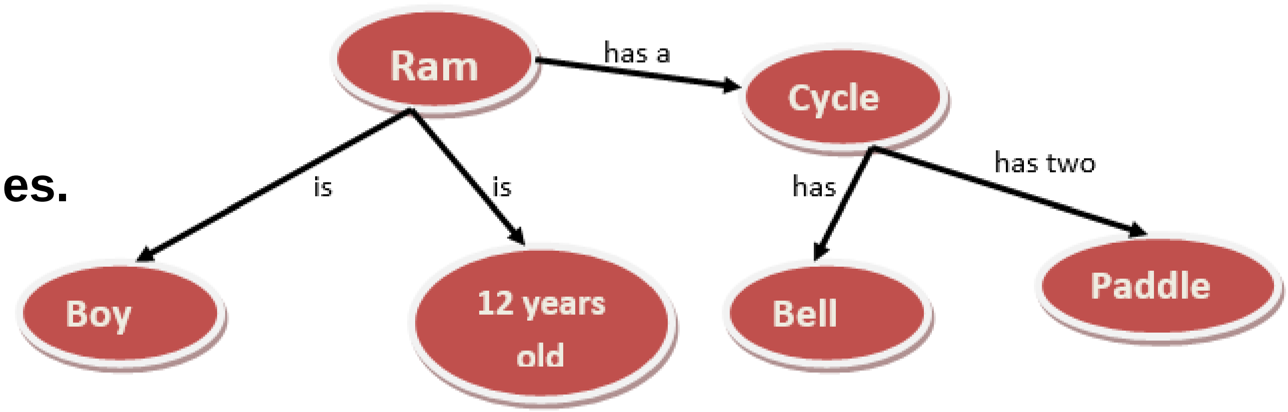
- The technique is based on storing the knowledge into the system in the form of a graph.

- Nodes of a graph represent the objects which exist in the real world, and the arrow represents the relationship between these objects.

- Such techniques show the connectivity of one object with another object.

Techniques used for KR

Ram has a cycle.
Ram is a boy.
Cycle has a bell.
Ram is 12 years old.
Cycle has two paddles.

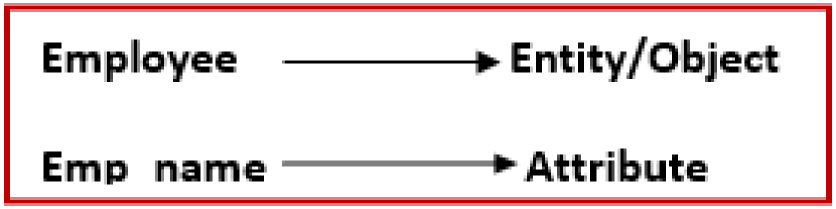


Techniques used for KR

Frames

- In this technique, the knowledge is stored via slots and fillers.

- As we have seen in DBMS, we store the information of an employee in the database with entities and attributes.



- Similarly, the Slots are the entities and Fillers are its attributes. They are together stored in a frame.

- So, whenever there is a requirement, the machine infers the necessary information to take the decision.

- For example, Tomy is a dog having one tail.

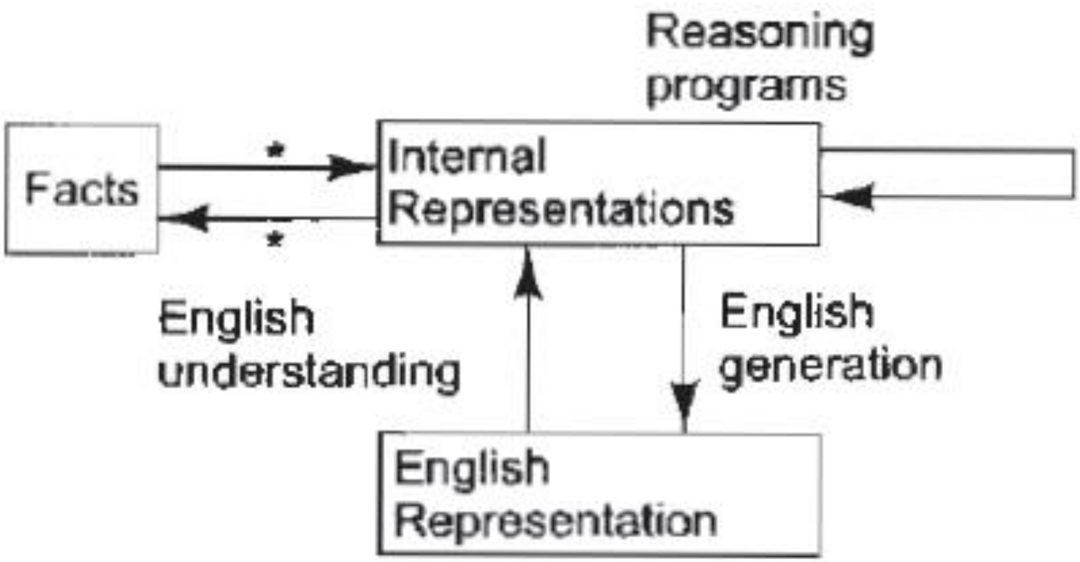
It can be framed as:
Tomy((Species (Value = Dog))
(Feature (Value = Tail)))

Techniques used for KR

Script:

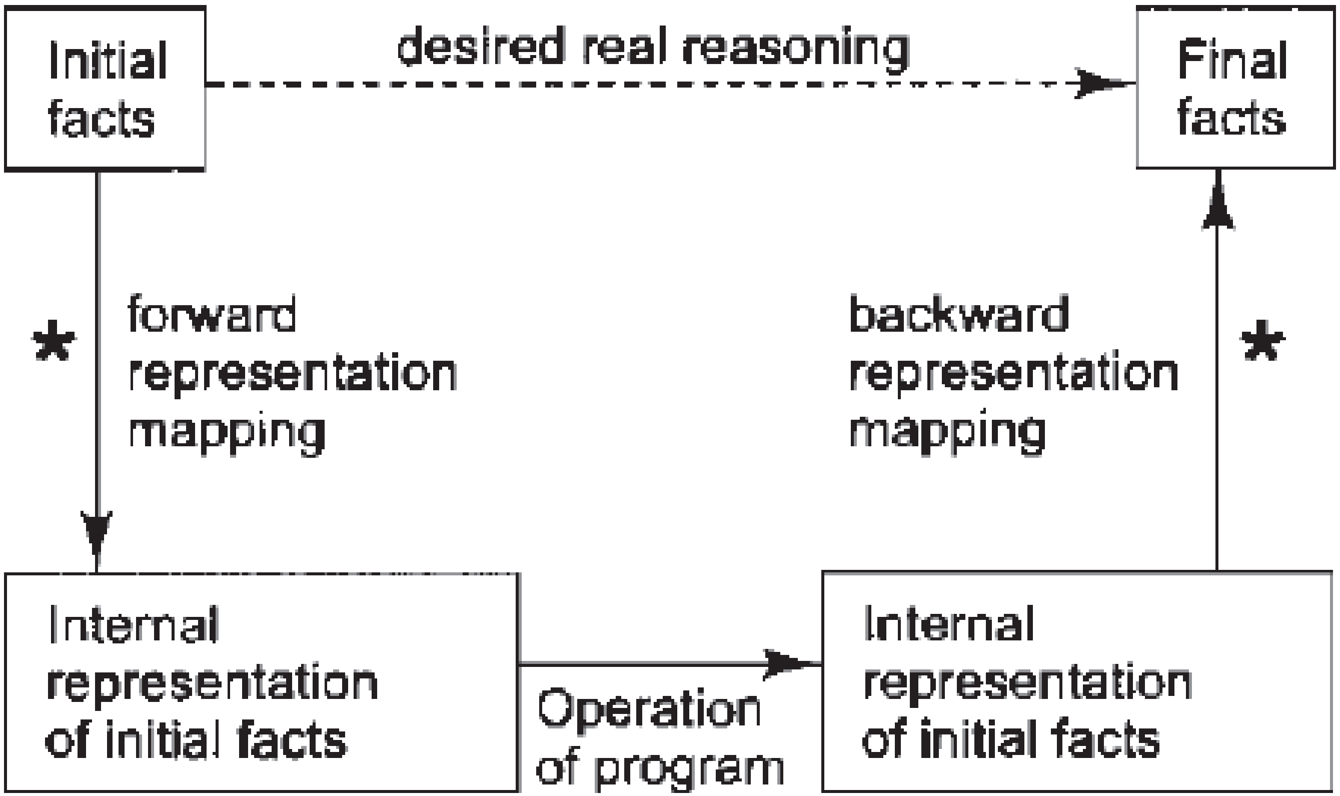
- It is an advanced technique over the Frames.
- Here, the information is stored in the form of a script.
- The script is stored in the system containing all the required information.
- The system infers the information from that script and solves the problem

Knowledge Representation and Mapping



- **Spot is a dog**
- **dog(spot)**
- **[∀]x : dog(x) -> hasatail(x),** logical expression for the fact that all dogs have tail
- New representation, **hasatail(Spot)**
- Using an appropriate backward mapping function the English sentence can be generated as **Spot has a tail**

Knowledge Representation and Mapping



Issues in KR

- **Attributes**
  - ✓ Are they basic?
  - ✓ Are they occuring frequently?
  - ✓ How are they properly represented?
  - ✓ E.g., ISA, INSTANCE

- **Relationship among attributes**
  - ✓ Inverse
  - ✓ Existence in a ISA hierarchy
  - ✓ Techniques for reasoning about value
  - ✓ Single valued attributes

- **Level of KR**
  - ✓ Use of primitives to represent knowledge
  - ✓ Can knowledge be broken down into a defined set of primitives
  - ✓ How such primitives help in KR

- **Object Representation**

- **How to access knowledge from repository**
  - ✓ Use of primitives to represent knowledge
  - ✓ Can knowledge be broken down into a defined set of primitives
  - ✓ How such primitives help in KR

Artificial Intelligence

Open Elective

Module2: Knowledge Representation: CH5

Dr. Santhi Natarajan
Associate Professor
Dept of AI and ML
BMSIT, Bangalore

KR: Simple Facts using Logic

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]**

KR: Simple Facts using Logic

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”

KR: Simple Facts using Logic

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	$(p \implies q) \iff (\sim p \vee q)$	$\sim(p \implies q) \iff (p \wedge \sim q)$

KR: Simple Facts using Logic

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	$\sim p \implies F$ $\therefore p$

KR: Simple Facts using Logic

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:

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

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

KR: Simple Facts using Logic

4. Distributive Rule:

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

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

5. De-Morgan's Rule:

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

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

6. Implication elimination:

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

KR: Simple Facts using Logic

7. Bidirectional Implication elimination:

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

8. Contrapositive rule:

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

9. Double Negation rule:

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

10. Absorption Rule:

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

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

KR: Simple Facts using Logic

11. Fundamental identities:

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

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

$P \vee T \Rightarrow P$

$P \vee F \Rightarrow P$

$P \vee \neg T \Rightarrow P$

$P \wedge F \Rightarrow F$

$P \wedge T \Rightarrow P$

KR: Simple Facts using Logic

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** .

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

KR: Simple Facts using Logic

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 **q** is true.

16. AND elimination:

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

$$P \wedge Q \rightarrow P$$

$$P \wedge Q \rightarrow Q$$

KR: Simple Facts using Logic

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:

$$P \rightarrow P \lor Q$$

$$Q \rightarrow P \lor Q$$

KR: Simple Facts using Logic

- 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"

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

KR: Simple Facts using Logic

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

KR: Simple Facts using Logic

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:

1. $(P \vee Q \vee R)$
2. $(P \vee Q)$
3. $(P \vee R)$
4. P

clauses

KR: Simple Facts using Logic

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 ($\wedge$)

KR: Simple Facts using Logic

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.

KR: Simple Facts using Logic

Conversion to CNF example

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

Solution:

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

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

Step 2: $\neg(\neg P \lor Q) \lor R$ ==> $(P \land \neg Q) \lor R$

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

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.

KR: Simple Facts using Logic

- 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" [$O \rightarrow H$] ==> $\neg O \vee H$
- 2. "If it is humid then it will rain". [$H \rightarrow R$] ==> $\neg H \vee R$
- 3. " It is Hot" [O] ==> O
-
- To prove: R.

KR: Simple Facts using Logic

- Let us assume “it will NOT rain” [$\neg R$]

- [$\neg R$] [$\neg H \vee R$]

$\neg H$ [$\neg O \vee H$]

$\neg O$ O

E [EMPTY CLAUSE / CONTRADICTION]

KR: Simple Facts using Logic

- 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

KR: Simple Facts using Logic

PREDICATE LOGIC

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

KR: Simple Facts using Logic

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)

KR: Simple Facts using Logic

Using Predicate Logic

1. Marcus was a man.

   man(Marcus)

2. Marcus was a Pompeian.

   Pompeian(Marcus)

KR: Simple Facts using Logic

- Quantifiers:
- 2 types:-

- <u>Universal quantifier (∀)</u>

- ∀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”

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

KR: Simple Facts using Logic

- 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)

KR: Simple Facts using Logic

Nested Quantifiers

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

- Ex: " everybody loves somebody "

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

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

$$\forall x: \neg like(x, garlic)$$

➢ **This can be also said as:**

"there does not exists someone who likes garlic"

$$\neg \exists x: like(x, garlic)$$

KR: Simple Facts using Logic

Using Predicate Logic

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.

KR: Simple Facts using Logic

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 Pompeians were either loyal to Caesar or hated him.
inclusive-or
$\forall$x: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar)
exclusive-or
$\forall$x: Roman(x) $\rightarrow$ (loyalto(x, Caesar) $\wedge$ $\neg$hate(x, Caesar)) $\vee$
($\neg$loyalto(x, Caesar) $\wedge$ hate(x, Caesar))

KR: Simple Facts using Logic

6. Every one is loyal to someone.

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

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

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

8. Marcus tried to assassinate Caesar.

tryassassinate(Marcus, Caesar)

Was Marcus loyal to Caesar?

man(Marcus)
ruler(Caesar)
tryassassinate(Marcus, Caesar)

$\Downarrow$

$\forall x: man(x) \rightarrow person(x)$
$\neg loyalto(Marcus, Caesar)$

KR: Simple Facts using 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).

KR: Simple Facts using Logic

Some more examples

- "all indoor games are easy"

  $\forall x$: indoor_game(x) → 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)→ king(x)}

KR: Simple Facts using Logic

- "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:

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

KR: Simple Facts using Logic

Computable Functions and Predicates

1. Marcus was a man.

2. Marcus was a Pompeian.

3. Marcus was born in 40 AD.

4. All men are mortal.

5. All Pompeians died when the volcano errupted in 79 AD.

6. No mortal lives longer than 150 years.

7.We are now in 2019 AD.

8. Alive means not dead.

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

KR: Simple Facts using Logic

1. Marcus was a man.
man(Marcus)

2. Marcus was a Pompeian.
Pompeian(Marcus)

3. Marcus was born in 40 AD.
Born(Marcus, 40)

4. All men are mortal.
$\forall$x: man(x) $\rightarrow$ mortal(x)

5. All Pompeians died when the volcano errupted in 79 AD.
Erupted(volcano, 79) $\wedge$ $\forall$x: [Pompeian (x) $\rightarrow$ Died (x , 79)]

6. No mortal lives longer than 150 years.
$\forall x: \forall t_1: \forall t_2: died(x, t_1) \wedge greater\text{-}than(t_2, -t_{1,150}) \rightarrow dead(x, t_2)$

KR: Simple Facts using Logic

7.We are now in 2019 AD.

now = 2008

8. Alive means not dead.

$\forall$**x:** $\forall$**t: [Alive (x, t)** $\rightarrow \neg$**dead(x, t)]** $\wedge$ **[dead(x, t)** $\rightarrow \neg$ **Alive (x, t)]**

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

$\forall$**x:** $\forall$**t_1:** $\forall$**t_2: died(x, t_1)** $\wedge$ **greater-than(t_2, t_1)** $\rightarrow$ **dead(x, t_2)**

KR: Simple Facts using Logic

Reasoning: Direct Proof

Is Marcus alive?

1. Pompeian(Marcus)

5. $\forall x$ Pompeian(x) $\Rightarrow$ died(x,79)
died(Marcus,79)

8. gt(now,79)
died(Marcus,79) $\wedge$ gt(now,79)

7. $\forall x\ \forall t1\ \forall t2$ died(x,t1) $\wedge$ gt(t2,t1) $\Rightarrow$ dead(x,t2)
dead(Marcus,now)

KR: Simple Facts using Logic

Reasoning: Proof by Contradiction

Is Marcus alive?

$\neg$ dead(Marcus,now)

$\forall x\ \forall t1\ \forall t2$ died(x,t1) $\wedge$ gt(t2,t1) $\Rightarrow$ dead(x,t2)

$\forall t1\ \neg$ [died(Marcus,t1) $\wedge$ gt(now,t1)]

$\forall t1\ \neg$ died(Marcus,t1) $\vee$ $\neg$ gt(now,t1)

died(Marcus,79)*

$\neg$ gt(now,79)

gt(now,79)

contradiction

KR: Resolution

- Resolution is an iterative process. At each step, two parent clauses are compared and resolved, yielding a new clause that is inferred from them. The new clause represents ways that the two parent clauses can interact with each other.
  - p
  - ¬p
- A *clausal sentence* is either a literal or a disjunction of literals. If *p* and *q* are logical constants, then the following are clausal sentences.
  - p
  - ¬p
  - ¬p ∨ q
- A *clause* is the set of literals in a clausal sentence. For example, the following sets are the clauses corresponding to the clausal sentences above.
  - {p}
  - {¬p}
  - {¬p, q}

KR: Resolution

- A *literal* is either an atomic sentence or a negation of an atomic sentence. For example, if *p* is a logical constant, the following sentences are both literals.
  - winter ∨ summer (TRUE)
  - ¬winter ∨ cold (TRUE)
- Resolution operates by taking two clauses, such that each contain the same literal that occurs in positive form in one clause and negative form in another clause.

- The resolvent is obtained by combining all of the literals of the two parent clauses except the ones that cancel.
  - Summer ∨ cold (RESOLVENT)
  - ¬winter, winter will produce EMPTY clause

- If a contradiction exists, then eventually it will be found. If no contradiction exists, it is possible that the procedure will never terminate.

KR: Resolution Procedure

1. Convert F to clause form: a set of clauses.

2. Negate S, convert it to clause form, and add it to your set of clauses.

3. Repeat until a contradiction or no progress
   a. Select two parent clauses.
   b. Produce their resolvent.
   c. If the resolvent = NIL, we are done.
   d. Else add the resolvent to the set of clauses.

KR: Resolution CNF

1. Eliminate $\rightarrow$.

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

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

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

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

KR: Resolution CNF

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

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

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

$\neg \forall x: P \equiv \exists x: \neg P$

$\neg \exists x: p \equiv \forall x: \neg P$

$\neg\neg P \equiv P$

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

$\forall$**x: [Roman (x)** $\land$ **(**$\neg$**Pompeian(x)** $\lor$ hate (x, Caesar))]

KR: Resolution CNF

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

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

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

$\forall x$: [[$\forall y$: 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)]]

KR: Resolution CNF

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

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

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

- $\forall x$: [[$\forall y$: 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

KR: Resolution CNF

5. Eliminate ∃ (Skolemization).

∃x: P(x) ≡ **P(c)** Skolem constant

∃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

∀**x:** ∃**y P(x, y)** ≡ ∀**x: P(x, f(x))** Skolem function

∃y: ∀x: leads (y , x)

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

∀x: leads (f(x) , x)

Where f(x) is skolem function.

KR: Resolution CNF

6. Drop $\forall$.

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

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

- After prefix dropped becomes,

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

KR: Resolution CNF

7. Convert the formula into a conjunction of disjuncts.

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

- Roman (x) $\vee$ ((hate (x , caesar) $\wedge$ $\neg$loyalto (x , caesar))
- Roman (x) $\vee$ ((hate (x , caesar) $\wedge$ $\neg$loyalto (x , caesar))

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

CLAUSE 1 (Roman (x) $\vee$ (hate (x , caesar)) $\wedge$

CLAUSE 2 (Roman (x) $\vee$ $\neg$loyalto (x , caesar))

8. Create a separate clause corresponding to each conjunct.

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

KR: Resolution Operations

- Note that the empty set {} is also a clause. It is equivalent to an empty disjunction and, therefore, is unsatisfiable. As we shall see, it is a particularly important special case.

- Implications (I):

  - $\varphi \Rightarrow \psi \;\longrightarrow\; \neg\varphi \vee \psi$
  - $\varphi \Leftarrow \psi \;\longrightarrow\; \varphi \vee \neg\psi$
  - $\varphi \Leftrightarrow \psi \;\longrightarrow\; (\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$

- Negations (N):

  - $\neg\neg\varphi \;\longrightarrow\; \varphi$
  - $\neg(\varphi \wedge \psi) \longrightarrow \neg\varphi \vee \neg\psi$
  - $\neg(\varphi \vee \psi) \longrightarrow \neg\varphi \wedge \neg\psi$

KR: Resolution Operations

- Distribution (D):
  - $\varphi \vee (\psi \wedge \chi) \rightarrow (\varphi \vee \psi) \wedge (\varphi \vee \chi)$
  - $(\varphi \wedge \psi) \vee \chi \rightarrow (\varphi \vee \chi) \wedge (\psi \vee \chi)$
  - $\varphi \vee (\varphi 1 \vee \ldots \vee \varphi n) \rightarrow \varphi \vee \varphi 1 \vee \ldots \vee \varphi n$
  - $(\varphi 1 \vee \ldots \vee \varphi n) \vee \varphi \rightarrow \varphi 1 \vee \ldots \vee \varphi n \vee \varphi$
  - $\varphi \wedge (\varphi 1 \wedge \ldots \wedge \varphi n) \rightarrow \varphi \wedge \varphi 1 \wedge \ldots \wedge \varphi n$
  - $(\varphi 1 \wedge \ldots \wedge \varphi n) \wedge \varphi \rightarrow \varphi 1 \wedge \ldots \wedge \varphi n \wedge \varphi$

- Operators (O):
  - $\varphi 1 \vee \ldots \vee \varphi n \rightarrow \{\varphi 1, \ldots, \varphi n\}$
  - $\varphi 1 \wedge \ldots \wedge \varphi n \rightarrow \{\varphi 1\}, \ldots, \{\varphi n\}$

KR: Resolution Example

- Consider the job of converting the sentence ($g \wedge (r \Rightarrow f)$) to clausal form

	$g \wedge (r \Rightarrow f)$
I	$g \wedge (\neg r \vee f)$
N	$g \wedge (\neg r \vee f)$
D	$g \wedge (\neg r \vee f)$
O	$\{g\}$
	$\{\neg r, f\}$

- Option II

	$\neg(g \wedge (r \Rightarrow f))$
I	$\neg(g \wedge (\neg r \vee f))$
N	$\neg g \vee \neg(\neg r \vee f)$
	$\neg g \vee (\neg\neg r \wedge \neg f)$
	$\neg g \vee (r \wedge \neg f)$
D	$(\neg g \vee r) \wedge (\neg g \vee \neg f)$
O	$\{\neg g, r\}$
	$\{\neg g, \neg f\}$

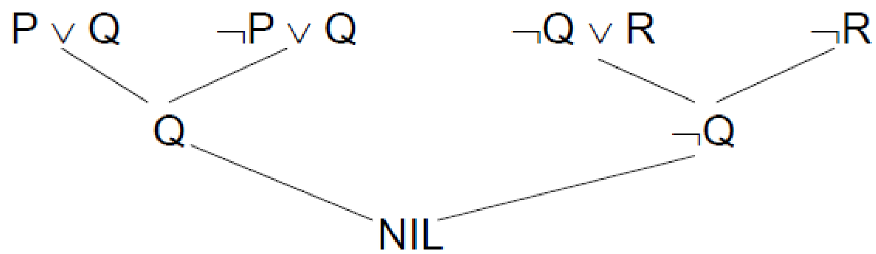
KR: Resolution Propositional

- Let C1 = L1 ∨L2 ∨… ∨Ln
- Let C2 = L1’ ∨L2’ ∨… ∨Ln’
- If C1 has a literal L and C2 has the opposite literal ¬L, they cancel each other and produce
- resolvent(C1,C2) = L1 ∨L2 ∨… ∨Ln ∨L1’ ∨L2’ ∨… ∨Ln’ with both L and ¬L removed
- If no 2 literals cancel, nothing is removed

KR: Resolution Propositional

Formulas: $P \vee Q$, $P \Rightarrow Q$, $Q \Rightarrow R$
Conjecture: R

Original Clauses: $\{P \vee Q, \neg P \vee Q, \neg Q \vee R, \neg R\}$



- Negation of conjecture: $\neg R$
- Clauses: $\{P \vee Q, \neg P \vee Q, \neg Q \vee R, \neg R\}$
- Resolvent($P \vee Q$, $\neg P \vee Q$) is Q. Add Q to clauses.
- Resolvent($\neg Q \vee R$, $\neg R$) is $\neg Q$. Add $\neg Q$ to clauses.
- Resolvent(Q, $\neg Q$) is NIL.
- The conjecture is proved.

KR: Resolution Unification

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

KR: Resolution Unification

- Used in predicate logic for resolution.

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

∀x: knows(John, x) → hates(John, x)
knows(John, Jane)
∀y: knows(y, Leonid)
∀y: knows(y, mother(y))
∀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

KR: Resolution Unification

- Used in predicate logic for resolution.

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

KR: Resolution Predicate Logic

- It is used as inference mechanism.

- Pre-processing steps:

1. Convert the given English sentence into predicate sentence.
2. Not all of these sentences will be in clausal form (CNF). If any sentence is not in clausal form then convert it into clausal form.
3. Give these sentences (clauses) as an input to resolution algorithm.

KR: Resolution Predicate Logic

Resolution algorithm steps:

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.

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

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.

KR: Resolution Predicate Logic

P(x) ∨ Q(x) R(x) ∨ ¬ **P(X)**

Q(x) ∨ R(x)

G(x) ¬ G(x)

E **[EMPTY CLAUSE]**

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.

KR: Resolution Predicate Logic

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

KR: Resolution Predicate Logic

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

KR: Resolution Predicate Logic

4. "Caesar was a ruler"

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

5. "all romans were either loyalto caesar or hated him"

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

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

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

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

KR: Resolution Predicate Logic

- "Every one is loyal to someone"

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

Let f(x3) be a skolem function then,

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

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

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

KR: Resolution Predicate Logic

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,

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

KR: Resolution Predicate Logic

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

$\neg$ man(x4) $\vee \neg$ ruler(f(x4)) $\vee$ $\neg$ tryassassinate(x4, f(x4)) $\vee$ $\neg$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)

KR: Resolution Predicate Logic

- 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)

KR: Resolution Predicate Logic

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)

KR: Resolution Predicate Logic

¬ 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.

KR: Resolution Predicate Logic

- 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]

KR: Resolution Predicate Logic

- Solution:

1. " anything anyone eats is called food."

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

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

$\Rightarrow$ **$\neg$ eats(x , y) $\vee$ food(y)** **(1)**

2. "Milka likes all kind of food"

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

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

$\Rightarrow$ **$\neg$ food(y1) $\vee$ like(milka , y1)** **(2)**

3. "Bread is a food"

food(bread) **(3)**

4. "Mango is a food"

food(mango) **(4)**

KR: Resolution Predicate Logic

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)

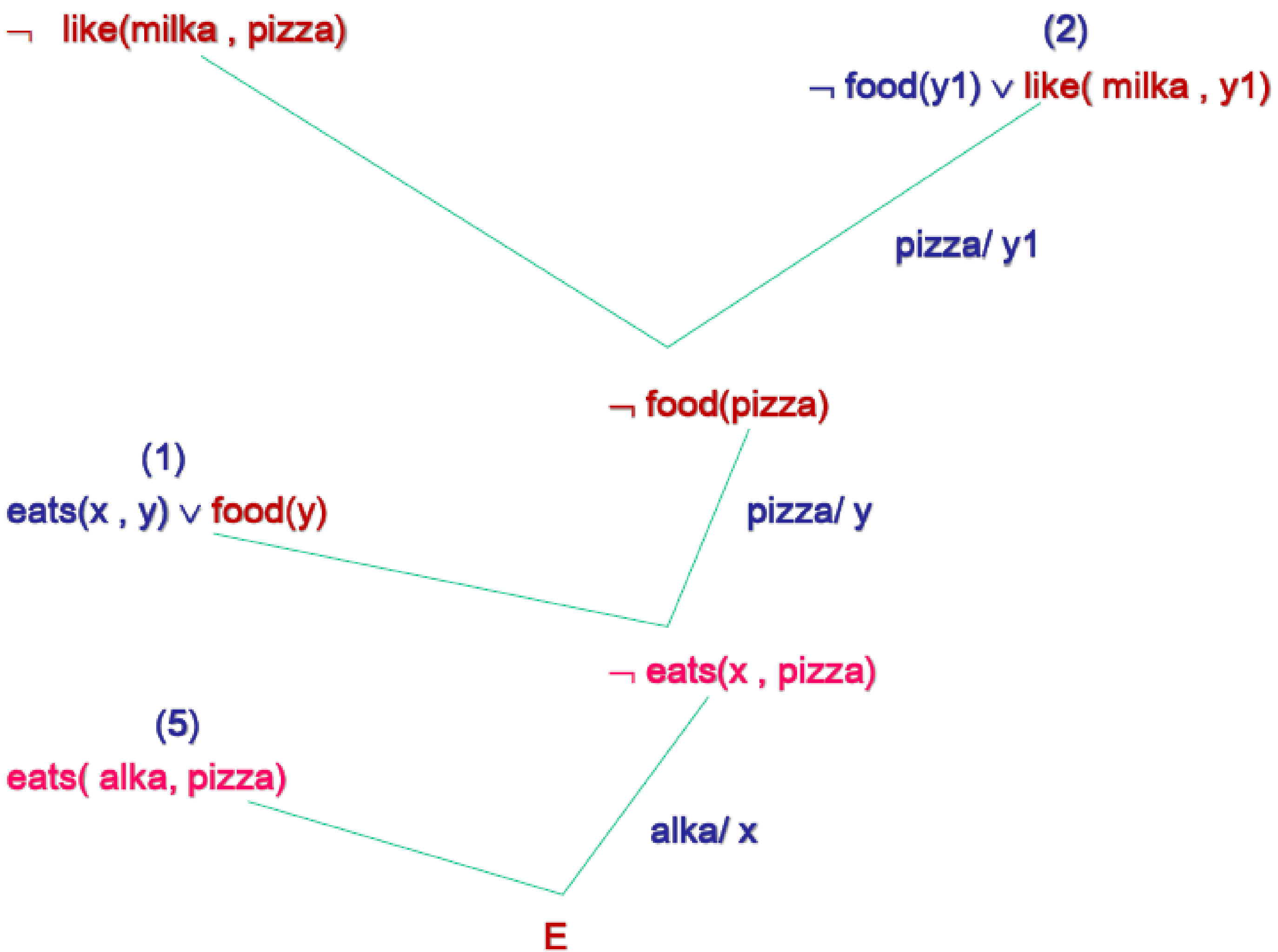
=> $\neg$ **eats(milka , x1)** $\vee$ **eats(alka, x1)** (6)

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

assume : “Milka **does not** like Pizza”

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

KR: Resolution Predicate Logic



Since ¬ like(milka , pizza) is contradiction like(milka , pizza) is true

KR: Resolution Predicate Logic

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

KR: Resolution Predicate Logic

- 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"

KR: Resolution Predicate Logic

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

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

KR: Resolution Predicate Logic

- 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 :

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

KR: Simple Facts using Logic

Different Logics

	Propositional Logic	Predicate Logic	Temporal (Modal) Logic
Atomic symbols	Concrete objects, AND, OR, NOT, IF-THEN.	Propositional logic + variables + quantifiers: for all, exists.	Predicate logic + temporal operators: always, eventually, until, ...
Examples of formalizable statements	My son is at home **and** my dad is not. It **either** rains **or** it does not. **If** the president sleeps, **then** a war cannot start.	**All** husbands cheat. If **all** colleges are bad, CMU is bad. **At least one** chinchilla is smarter than at **least one** human.	**Eventually** all humans will die. I **always** tell ducks not to panic.

Procedural Versus Declarative

- Declarative knowledge is defined as the factual information stored in memory and known to be static in nature. Also known as descriptive knowledge, propositional knowledge, etc

- It is the part of knowledge which describes how things are.

- Things/events/processes, their attributes, and the relations between these things/events/processes and their attributes define the domain of declarative knowledge.

- Procedural knowledge is the knowledge of how to perform, or how to operate. Names such as know-how are also given.

- It is said that one becomes more skilled in problem solving when he relies more on procedural knowledge than declarative knowledge.

- It embeds control information in the knowledge base, only to the extent that the interpreter for the knowledge base recognizes the control information.

Procedural Versus Declarative

BASIS FOR COMPARISON	PROCEDURAL KNOWLEDGE	DECLARATIVE KNOWLEDGE
Basic	Includes the knowledge of how a particular thing can be accomplished.	Includes the basic knowledge about something.
Alternate name	Interpretive knowledge	Descriptive knowledge
Stated by	Direct application to the task and difficult to articulate formally.	Declarative sentences and easily articulated.
Popularity	Less common	Generally used
Ease of sharing the knowledge	Hard to communicate	Can be easily shared, copied, processed and stored.
Taken from	Experience, action, and subjective insight.	Artifact of some type as a principle, procedure, process and concepts.
Nature	Process oriented	Data-oriented
Represented by	Set of rules	Production systems
Feature	Debugging is difficult	Validation is quite simple

Procedural Versus Declarative

Procedural knowledge	**Declarative knowledge**
• high efficiency • low modifiability • low cognitive adequacy (better for knowledge engineers)	• higher level of abstraction • suitable for independent facts • good modifiability • good readability • good cognitive matching (better for domain experts and end-users) •low computational efficiency

Procedural Versus Declarative

man(Marcus)

man(Ceasar)

Person(Cleopatra)

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

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

Y = Marcus

Y = Ceasar

Y = Cleopatra (DFS)

man(Marcus)

man(Ceasar)

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

Person(Cleopatra)

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

Y = Marcus (DFS)

Y = Ceasar

Y = Cleopatra

man(Marcus)

man(Ceasar)

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

Person(Cleopatra)

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

Y = Marcus

Y = Ceasar (DFS, Last to First)

Y = Cleopatra