AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER – 4
USING PREDICATE LOGIC

Subject : AI Code : 2180703	Prepared By: Asst. Prof. Twinkal Panchal (CSE Department, ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Representing Simple Facts In Logic

First-Order Logic

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, ...
- **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of, ...

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression.
  1. **Universal Quantifier, (for all, everyone, everything)**
  2. **Existential quantifier, (for some, at least one).**

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

1. **Universal Quantifier**

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.
- The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.
- Note: In universal quantifier we use implication "→".
- If x is a variable, then ∀x is read as:
- **For all x**
- **For each x**
- **For every x.**

Ex-

All man drink coffee.

∀x man(x) → drink (x, coffee).

2. Existential Quantifier

- Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.
- It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.
- If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:
- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Ex-

Some boys are intelligent.

$\exists x$: boys(x) ∧ intelligent(x)

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

1. Marcus was a man.

   Man(Marcus).

2. Marcus was a Pompeian.

   Pompeian(Marcus).

3. All Pompeians were Romans.

   $\forall x$: Pompeian(x) - > Roman(x).

4. Caesar was a ruler.

   Ruler(Caesar).

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

   $\forall x$:Roman(y) -> (LoyalTo(x,Caesar) $\vee$ Hate(x,Caesar))

6. Everyone is loyal to someone .

$\forall x$:-> 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)) -> $\neg$LoyalTo(x,y)

8. Marcus tried to assassinate Caesar .

TryAssassinate(Marcus, Caesar).

Representing Instance and Isa Relationships

- Logic statements, containing subject, predicate, and object, were explained. Also stated, two important attributes "instance" and "isa", in a hierarchical structure.

- Attributes " **IsA** " and " **Instance** " support property inheritance and play important role in knowledge representation.

- The ways these two attributes "instance" and "isa", are logically expressed are shown in the example below :

Ex-

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

A simple sentence like **"Joe is a musician"**

- Here **"is a"** (called **IsA**) is a way of expressing what logically is called a class-instance relationship between the subjects represented by the terms "**Joe**" and "**musician**".

- "**Joe**" is an instance of the class of things called "**musician**". "**Joe**" plays the role of instance,

- "**musician**" plays the role of class in that sentence.

Computable Functions and Predicates

- It may be necessary to compute functions as part of a fact. In these cases a computable predicate is used.
- A computable predicate may include computable functions such as +, -, *, etc.
- For example, gt(x-y,10) →bigger(x) contains the computable predicate gt which performs the greater than function.

Ex-

1. Marcus was a man.

   man(Marcus).

2. Marcus was Pompeian.

   Pompeian(Marcus).

3. Marcus was born in 40 A.D.

born(Marcus,40).

4. All men are mortal.

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

5. All Pompeians died when the volcano erupted in 79 A.D.

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

6. No mortal lives longer than 150 years.

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

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

7. It is now 1991.

$$now=1991.$$

8. Alive means not dead.

$$\forall x \quad \forall t: \quad [alive(x,t) \quad \rightarrow \quad \sim dead(x,t)] \quad \wedge \quad [\sim dead(x,t) \rightarrow alive(x,t)]$$

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

$$\forall x \ \forall t1 \ \forall t2: died(x,t1) \wedge gt(t2,t1) \rightarrow dead(x,t2).$$

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Ex **- "Is Marcus alive now?".**

~alive(Marcus,now)

↓ 8

~[~dead(Marcus,now)]

↓ negation operation

dead(Marcus,now)

↓ 9

died(Marcus,t1) /\ gt(now,t1)

↓ 5

erupted(volcano,79) /\ Pompeian(Marcus) /\ gt(now,79)

↓fact, 2

gt(now,79)

↓

gt(1991,79)

↓ compute gt

nil

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Resolution

- Resolution proves facts and answers queries by refutation. This involves assuming the fact/query is untrue and reaching a contradiction which indicates that the opposite must be true. The wffs must firstly be firstly converted to clause form before using resolution.

Algorithm: Converting wffs to Clause Form

1. Remove all implies, i.e. $\rightarrow$ by applying the following: $a \rightarrow b$ is equivalent to $\sim a \vee b$.

2. Use the following rules to reduce the scope of each negation operator to a single term:
   - $\sim(\sim a) = a$
   - $\sim(a \wedge b) = \sim a \vee \sim b$
   - $\sim(a \vee b) = \sim a \wedge \sim b$

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

$\sim\forall x: p(x) = \exists x: \sim p(x)$
$\sim\exists x: p(x) = \forall x: \sim p(x)$

3. Each quantifier must be linked to a unique variable. For example, consider $\forall x: p(x) \vee \forall x: q(x)$. In this both quantifiers are using the same variable and one must changed to another variable: $\forall x: p(x) \vee \forall y: q(y)$.

4. Move all quantifiers, in order, to the left of each wff.

5. Remove existential quantifiers by using Skolem constants or functions. For example, $\exists x: p(x)$ becomes $p(s1)$ and $\forall x\ \exists y: q(x,y)$ is replaced with $\forall x: q(s2(x), x)$.

6. Drop the quantifier prefix.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

7. Apply the associative property of disjunctions: $a \lor (b \lor c) = (a \lor b) \lor c$ and remove brackets giving $a \lor b \lor c$.

8. Remove all disjunctions of conjunctions from predicates, i.e. create conjunctions of disjunctions instead, by applying the following rule iteratively: $(a \land b) \lor c = (a \lor c) \land (b \lor c)$.

9. Create a separate clause for each conjunction.

10. Rename variables in the clauses resulting from step 9. to ensure that no two clauses refer to the same variable.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Algorithm: Resolution

1. Convert the wffs to clause form.

2. Add the fact (or query) P to be proved to the set of clauses:

    i. Negate P.

    ii. Convert negated P to clause form.

    iii. Add the result of ii to the set of clauses.

3. Repeat

    i. Select two clauses.

    ii. Resolve the clauses using unification.

    iii. If the resolvent clause is the empty clause, then a contradiction has been reached. If not add the resolvent to the set of clauses.

Consider the following wffs:

1. man(Marcus).

    man(Marcus).

2. Pompeian(Marcus).

    Pompeian(Marcus).

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

    ~Pompeian(x) $\vee$ Roman(x).

4. ruler(Caesar).

    ruler(Caesar).

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

    ~Roman(x1) $\vee$ loyalto(x1,Caesar) $\vee$ hate(x1,Caesar).

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

loyalto(x2,s1(x2))

7. $\forall x\ \forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x,y) $\rightarrow$ ~loyalto(x,y)

~person(x3) $\vee$ ~ruler(y) $\vee$ ~tryassassinate(x3,y) $\vee$ ~loyalto(x3,y)

8. tryassassinate(Marcus,Caesar).

tryassassinate(Marcus, Caesar).

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

~man(x4) $\vee$ person(x4)

- we want to prove that Marcus hates Caesar.
- We firstly convert this to a wff: hate(Marcus,Caesar).
- The wff is then negated and converted to clause form: ~hate(Marcus,Caesar).
- This clause is added to the set of clauses and the resolution is algorithm is applied:

$\sim hate(Marcus,Caesar)$

$\downarrow 5$

$\sim Roman(Marcus) \lor loyalto(x1,Caesar)$

$\downarrow 3$

$\sim Pompeian(Marcus) \lor loyalto(Marcus,Caesar)$

$\downarrow 2$

$loyalto(Marcus,Caesar)$

$\downarrow 7$

$\sim person(Marcus) \lor \sim ruler(Caesar) \lor \sim tryassassinate(Marcus,Caesar)$

$\downarrow 4$

$\sim person(Marcus) \lor \sim tryassassinate(Marcus,Caesar)$

$\downarrow 8$

$\sim person(Marcus)$

$\downarrow 9$

$\sim man(Marcus)$

$\downarrow 1$

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY