Knowledge representation formalisms (techniques) **LECTURE 4**

Different types of knowledge require different types of representation.
◊ **Predicate logic :**Predicate is a function may be TRUE for some arguments, and FALSE for others.
◊ **Semantic networks :**A semantic net is just a graph, where the nodes represent concepts, and the arcs represent binary relationships between concepts.
◊ **Frames and scripts :** A frame is a data structure that typically consists of : Frame name, Slot-filler (relations target), Pointers (links) to other Frames, Instantiation Procedure (inheritance, default, consistency). The Scripts are linked sentences using frame-like structures; e.g., a record of sequence of events for a given type of occurrence.
◊ **Production rules :** Consists of a set of rules about behavior; a production consists two parts: a precondition (or IF) and an action (or THEN); if a production's precondition matches the current state of the world, then the production is said to be triggered.

Forward and Backward Representation

Initial facts ---------forward representation mapping --------**internal representation** --- operated by program ---------**English representation** --------backward representation -----**Final facts.**

Desired reasoning is **Initial facts** -----to---- **Final Facts** this indicates the abstract reasoning process that a program is intended to model. The other links in between indicates the **concrete reasoning** process that the program performs.

Examples of predicate logic statements :
1. "Rex" is a name of a dog : dog (Rex)
2. All dogs belong to the class of animals : $\forall$ x : dog (x) $\rightarrow$ animal(x)
3. All animals either live on land or in water : $\forall$ x : animal(x) $\rightarrow$ live (x, land) V live (x, water)
From these three statements we can infer that : **" Rex lives either on land or on water."**

Issues in Knowledge Representation

The fundamental goal of Knowledge Representation is to facilitate inferencing (conclusions) from knowledge. The issues that arise while using KR techniques are many. Some of these are:-
◊ **Important Attributes** : Any attribute of objects so basic that they occur in almost every problem domain ?
◊ **Relationship among attributes**: Any important relationship that exists among object attributes ?
◊ **Choosing Granularity** : At what level of detail should the knowledge be represented ? Should there be a small number or should there be a large number of low-level primitives or High-level facts. High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.
◊ **Set of objects** : How sets of objects be represented ?
◊ **Finding Right structure** : Given a large amount of knowledge stored, how can relevant parts be accessed ?

Example of Granularity:
Suppose we are interested in following fact: John spotted Sue.
This could be represented as : Spotted (agent(John), object (Sue))
Such a representation would make it easy to answer questions such are :- Who spotted Sue ?

Suppose we want to know - Did John see Sue ?
Given only one fact, we cannot discover that answer.
We can add other facts, such as : Spotted (x , y) → saw (x , y)
We can now infer the answer to the question.

Knowledge representation Using Predicate Logic
How knowledge can be represented as "symbol structures" that characterize bits of knowledge about objects, concepts, facts, rules, strategies;

Examples :

- "red" represents color red;
- "car" represents my car ;
- "red(car)" represents fact that my car is red.

Assumptions about Knowledge Representation (KR) :

1. Intelligent Behavior can be achieved by manipulation of symbol structures.
2. KR languages are designed to facilitate operations over symbol structures, have precise syntax and semantics;
   - **Syntax** tells which expression is legal ?, e.g., red1(car1), red1 car1, car1(red1), red1(car1 & car2) ?;
   - **Semantic** tells what an expression means ? e.g., property "dark red" applies to my car.
3. Make Inferences, draw new conclusions from existing facts.

To satisfy these assumptions about KR, we need formal notation that allow automated inference and problem solving. One popular choice is use of logic.

LOGIC

Logic is concerned with the truth of statements about the world. It is a language for reasoning, a collection of rules used while doing logical reasoning. Each statement is either TRUE or FALSE.
Logic includes : Syntax , Semantics and Inference Procedure.
Logic reasoning is the process of drawing conclusions from premises using rules of inferences. The study of logic is divided into formal and informal logic. The **formal** logic is sometimes called symbolic logic.

Symbolic logic is the study of symbolic abstractions (construct) that capture the formal features of logical inference by a formal system. **Formal system** consists of two components, a formal language plus a set of inference rules. The formal system has axioms.

◊ **Inference Procedure :** Specifies methods for computing new sentences from the existing sentences.
Facts : are claims about the world that are True or False.
Representation : is an expression (sentence), stands for the objects and relations.
Sentences : can be encoded in a computer program.

Logics are of different types : Propositional logic, Predicate logic, Temporal logic, Modal logic, Description logic. The fundamental types in AI are:- *Predicate Logic* is the study of individuals and their properties.
Propositional Logic is the study of statements and their connectivity.

Propositions are “sentences” , either true or false but not both.

- A sentence is smallest unit in propositional logic.
- If proposition is true, then truth value is "true" .
- If proposition is false, then truth value is "false" .

Example :

Sentence	**Truth value**	**Proposition (Y/N)**
"2 + 5 = 5"	"false"	Yes
"Grass is green"	"true"	Yes
"x > 2" where x is variable	-	No

the truth values of a group of propositions, can be either a tautology, contradiction, contingency, antecedent,consequent. They form argument where one proposition claims to follow logically other proposition

◊ **Tautologies -** A proposition that is always true is called a "tautology". e.g (P v ¬P) is always true regardless of the truth value of the proposition P.

◊ **Contradictions** : A proposition that is always false is called a "contradiction". e.g (P ∧ ¬P) is always false regardless of the truth value of the proposition P.

◊ **Contingencies** A proposition is called a "contingency" , if that proposition is neither a tautology nor a contradiction . e.g., (P v Q) is a contingency.

◊ **Antecedent**, **Consequent** These two are parts of conditional statements. In the conditional statements, p → q , the 1st statement or "if - clause" (here p) is called antecedent , 2nd statement or "then - clause" (here q) is called consequent.

Argument

An argument is a demonstration or a proof of some statement. Example : "That bird is a crow; therefore, it's black."

Any argument can be expressed as a compound statement. In logic, an argument is a set of one or more meaningful declarative sentences (or "propositions") known as the premises along with another meaningful declarative sentence (or "proposition") known as the conclusion.

Premise is a proposition which gives reasons, grounds, or evidence for accepting some other proposition, called the conclusion.

Conclusion is a proposition, which is purported to be established on the basis of other propositions.

Every argument has a corresponding conditional, and every implication statement has a corresponding argument. Because the corresponding conditional of an argument is a statement, it is therefore either a tautology, or a contradiction, or a contingency.

‡ An argument is **valid** "if and only if" its corresponding conditional is a tautology.

‡ Two statements are **consistent** "if and only if" their conjunction is not a contradiction.

‡ Two statements are logically **equivalent** "if and only if" their truth table columns are identical; "if and only if" the statement of their equivalence using " ≡ " is a tautology.

Ex: Derive the truth table for- ((((a ∧ b) ∧ c) ∨ ((¬ a ∧ b) ∧ c)) ∨ ((a ∧ ¬ b) ∧ c))

Representing “ IsA ” and “ Instance ” Relationships

■ Example : A simple sentence like "Joe is a musician"

◊ Here "is a" (called IsA) is a way of expressing what logically is called a represented class-instance relationship between the subjects 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.
The format is [Instance] IsA [Class]

Resolution
Resolution is a procedure used in proving that arguments which are expressible in predicate logic are correct. Resolution is a procedure that produces proofs by disproof or contradiction.
Resolution lead to refute a theorem-proving technique for sentences in propositional logic and first-order logic.
− Resolution is a rule of inference.
− Resolution is a computerized theorem prover.
− Resolution is so far only defined for Propositional Logic. The strategy is that the Resolution techniques of Propositional logic be adopted predicate logic.

Logic Programming
Logic programming offers a philosophy for specifying a computation in terms of logical relations between entities.
− logic program is a collection of logic statements.
− programmer describes all relevant logical relationships between the various entities.
− computation determines whether or not, a particular conclusion follows from those logical statements.

Characteristics of Logic program
Logic program is characterized by set of relations and inferences.
- Program consists of a set of axioms and a goal statement.
- Rules of inference determine whether the axioms are sufficient to ensure the truth of the goal statement.
- Execution of a logic program corresponds to the construction of a proof of the goal statement from the axioms.
- Programmer specify basic logical relationships, does not specify the manner in which inference rules are applied.

Thus **Logic + Control = Algorithms**

Examples of Logic Statements
- Statement A grand-parent is a parent of a parent.
- Statement expressed in more closely related logic terms as - A person is a grand-parent if she/he has a child and that child is a parent.
- Statement expressed in first order logic as - (for all) x: grandparent (x, y) :- parent (x, z), parent (z, y)
- read as x is the grandparent of y if x is a parent of z and z is a parent of y

Logic Programming Language
A programming language includes :
− the syntax
− the semantics of programs and
− the computational model.
There are many ways of organizing computations. The most familiar paradigm is procedural. The

program specifies a computation by saying "how" it is to be performed. FORTRAN, C, and Object-oriented languages fall under this general approach.
Another paradigm is **declarative**. The program specifies a computation by giving the properties of a correct answer. Prolog and logic data language (LDL) are examples of declarative languages, emphasize the logical properties of a computation.
Prolog and LDL are called logic programming languages. PROLOG (PROgramming LOGic) is the most popular Logic programming language of Artificial Intelligence (AI). It became popular with AI researchers, who know more about "what" and "how" intelligent behavior is achieved.

Syntax and Terminology
Data components are collection of data objects that follow hierarchy. Data object of any kind is also called terms. A **term** is a constant, a variable , or a compound. Simple data objects are not decomposable; e.g. atoms , numbers, constants, variables.

Syntax distinguishes the data objects, hence no need for declaring them. Structured data object are made of several components.

Atoms are:-
- a string of special characters such as: + - * / \ = ^ < > : ~ # $ & ! ; [] {} a lower-case letter, possibly followed by other letters of either case, digits, and underscore character. following are also atoms

Structured Data Objects : General Structures , Special Structures

◊ *General Structures* - a structured term is syntactically formed by a functor and a list of arguments.
- functor is an atom.
- list of arguments appear between parentheses.
- arguments are separated by a comma.
- each argument is a term (i.e., any Prolog data object).
- the number of arguments of a structured term is called its **arity**. e.g. greaterThan(9, 6), f(a, g(b, c), h(d)), plus(2, 3, 5)

◊ *Special Structures*
- In Prolog an ordered collection of terms is called a *list* .
- Lists are structured terms and Prolog offers a convenient notation to represent them:

* Empty list is denoted by the atom [ ].
* Non-empty list carries element(s) between square brackets, separating elements by comma.
e.g. [bach, bee], [apples, oranges, grapes]

Program Components

A Prolog program is a collection of predicates or rules. A predicate establishes a relationships between objects.
Clause, Predicate, Sentence, Subject
A Clause is a collection of grammatically-related words. Predicate is composed of one or more clauses. Clauses are the building blocks of sentences; every sentence contains one or more clauses.
A Complete Sentence has two parts: **subject** and **predicate**. **subject** is what (or whom) the sentence is about. **predicate** tells something about the subject.

Example 1 : "cows eat grass". It is a clause, because it contains the subject "cows" and the predicate “eat grass”
Example 2 "cows eating grass are visible from highway" This is a complete clause. the subject "cows eating grass" the predicate thought and “are visible form the highway” makes a complete thought.

The general form of clauses is <left-hand-side> :- <right-hand-side>. where LHS is a single goal called "**goal**" and RHS is composed of one or more goals, separated by commas, called "sub-goals" of the goal on left-hand side.
The symbol " :- " is pronounced as "it is the case" or "*such that*"
Example :grand_parent (X, Z) :- parent(X, Y), parent(Y, Z).
parent (X, Y) :- mother(X, Y).
parent (X, Y) :- father(X, Y).

A **clause** specifies the conditional truth of the goal on the LHS; goal on LHS is assumed to be true if the sub-goals on RHS are all true. A predicate is true if at least one of its clauses is true.

Programming Paradigms : Models of Computation
A complete description of a programming language includes the computational model, syntax, semantics and pragamatic considerations that that shape the language.

Models of Computation :

A computational model is a collection of values and operations, while computation is the application of a sequence of operations to a value to yield another value.

There are three basic computational models :
(a) Imperative - The Imperative model of computation, consists of a state and an operation of assignment which is used to modify the state. Programs consist of sequences of commands. Computations are changes in the state.
Example : Linear function - A linear function y = 2x + 3 can be written as Y := 2 ∗ X + 3

(b) Functional - The Functional model of computation, consists of a set of values, functions, and the operation of functions. The functions may be named and composed with other functions. It can take other functions as arguments and return results. Example 1 : Linear function A linear function y = 2x + 3 can be defined as: f (x) = 2 ∗ x + 3. The notations and methods form the base upon which problem solving methodologies rest.

(c) Logic.- The logic model of computation is based on relations and logical inference. A linear function y = 2x + 3 can be represented as : f (X , Y) if Y is 2 * X+3. Here the *function* represents the relation between X and Y.
Logic model formalizes the reasoning process. It is related to relational data bases and expert systems.

In addition to these, there are two programming paradigms :
(a) concurrent
(b) object-oriented programming .

Forward versus Backward Reasoning

Rule-Based system architecture consists a set of rules, a set of facts, and an inference engine. The need is to find what new facts can be derived.
Given a set of rules, there are essentially two ways to generate new knowledge: one, forward chaining and the other, backward chaining.

■ Forward chaining : also called data driven. It starts with the facts, and sees what rules apply. e.g if hot and Smoky THEN fire. If alarm beeps then add smoky. If fire then add switch_on_sprinklers.
■ Backward chaining : also called goal driven. It starts with something to find out, and looks for rules that will help in answering it. e.g if hot and Smoky THEN fire. If alarm beeps then add smoky. If fire then add switch_on_sprinklers. What if it is hot and alarm beeps do I switch on the sprinklers?

Conflict Resolution Strategy

Conflict set is the set of rules that have their conditions satisfied by working memory elements. Conflict resolution normally selects a single rule to fire. The popular conflict resolution mechanisms are : Refractory, Recency, Specificity.
◊ **Refractory -** a rule should not be allowed to fire more than once on the same data. discard executed rules from the conflict set. prevents undesired loops.
◊ **Recency -** rank instantiations in terms of the recency of the elements in the premise of the rule. rules which use more recent data are preferred. working memory elements are time-tagged indicating at what cycle each fact was added to working memory.
◊ **Specificity -** rules which have a greater number of conditions and are therefore more difficult to satisfy, are preferred to more general rules with fewer conditions. more specific rules are ‘better’ because they take more of the data into account.

Control Knowledge

An algorithm consists of : logic component, that specifies the knowledge to be used in solving problems, and control component, that determines the problem-solving strategies by means of which that knowledge is used.
Algorithm = logic + Control
The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency.
An algorithm may be formulated in different ways, producing same behavior. One formulation, may have a clear statement in logic component but employ a sophisticated problem solving strategy in the control component.
The other formulation may have a complicated logic component but employ a simple problem-solving strategy. The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm and therefore without changing the meaning of the algorithm.
The trend in databases is towards the separation of logic and control. The programming languages today do not distinguish between them. The programmer specifies both logic and control in a single language. The execution mechanism exercises only the most rudimentary problem-solving capabilities. Computer programs will be more often correct, more easily improved, and more readily adapted to new problems when programming languages separate logic and control, and when execution mechanisms provide more powerful problem-solving facilities of the kind provided by intelligent theorem-proving systems.