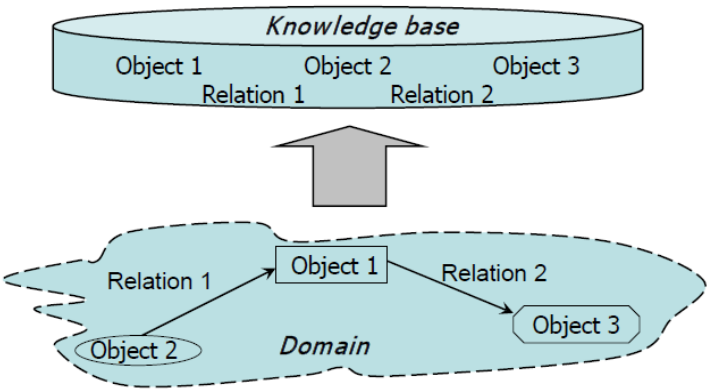
INTELLIGENT AGENTS

VI SEMESTER

UNIT- II

2.1 Knowledge Representation:

Knowledge representation is the method used to encode **knowledge** in an intelligent system's knowledge base. The object of knowledge representation is to express knowledge in computer-tractable form such that it can be used to help intelligent system perform well. A knowledge base is an integral part of any knowledge-based intelligent system. It maps objects and relationships of the real world to computational objects and relationships.



2.1.1 But What is Knowledge?

Knowledge is an abstract term that attempts to capture an individual's understanding of a given subject. In the world of intelligent systems the domain-specific knowledge is captured. Domain is a well-focused subject area. Cognitive psychologists have formed a number of theories to explain how humans solve problems. This work uncovered the types of knowledge humans commonly use, how they mentally organize this knowledge and how they use it efficiently to solve a problem.

2.1.2 Types of Knowledge:

Declarative knowledge	Concepts Facts Objects	Describes what is known about a problem. This includes simple statements that are asserted to be either true or false. This also includes a list of statements that more fully describes some object or concept (object-attribute-value triplet).
Procedural knowledge	Rules Strategies Agendas Procedures	Describes how a problem is solved. This type of knowledge provides direction on how to do something.
Heuristic Knowledge	Rules of Thumb	Describes a rule-of-thumb that guides the reasoning process. Heuristic knowledge is often called shallow

		knowledge. It is empirical and represents the knowledge compiled by an expert through the experience of solving past problems.
Meta Knowledge	Knowledge about the other types of knowledge and how to use them	Describes knowledge about knowledge. This type of knowledge is used to pick other knowledge that is best suited for solving a problem. Experts use this type of knowledge to enhance the efficiency of problem solving by directing their reasoning in the most promising area
Structural Knowledge	Rule sets Concept relationships Concept to object relationships	Describes knowledge structures. This type of knowledge describes an expert's overall mental model of the problem. The expert's mental model of concepts, sub concepts and objects is typical of this type of knowledge

2.1.3 Four Schemes of Knowledge Representation:

1. **Logical Schemes**
   * Predicate Calculus
   * Propositional Calculus
2. **Networked Schemes**
   * Semantic nets
   * Conceptual Graphs
3. **Procedural Schemes**
   * If .. Then else rules
4. **Structured Rules**
   * Scripts
   * Frames

http://bit.ly/2SybvTF

2.2 Rules of Inference and Formal Proofs:

Proofs in mathematics are valid arguments that establish the truth of mathematical statements. An argument is a sequence of statements that end with a conclusion. The argument is valid if the conclusion (final statement) follows from the truth of the preceding statements (premises). **Rules of inference are templates for building valid arguments.** Proving useful theorems using formal proofs would result in long and tedious proofs, where every single logical step must be provided. Proofs used for human consumption (rather than for automated derivations by the computer) are usually informal proofs, where steps are combined or skipped, axioms or rules of inference are not explicitly provided.

The rules of inference are the essential building block in the construction of valid arguments.

1. Propositional Logic
   - **(a)** Inference Rules
2. Predicate Logic
   - **(b)** Inference rules for propositional logic plus additional inference rules to handle variables and quantifiers

2.2.1 Propositional Logic:

An argument in propositional logic is a sequence of propositions. All but the final proposition are called premises. The last statement is the conclusion. The argument is valid if the premises imply the conclusion. An argument form is an argument that is valid no matter what propositions are substituted into its propositional variables. If the premises are p1 ,p2, …,pn and the conclusion is q then (p1 ∧ p2 ∧ … ∧ pn) → q is a **tautology**. Inference rules are all argument simple argument forms that will be used to construct more complex argument forms.

2.2.2 Rules of Inference for Propositional Logic:

NAME	INFERENCE RULES	TAUTOLOGY
Modus Ponens (mode that affirms)	p p → q _____ ∴ q	p ∧ (p → q)) → q
Modus Tollens (mode that denies)	~q p → q _____ ∴ ~p	(¬q ∧ (p → q)) → ¬p
Hypothetical Syllogism	p → q q → r _____ ∴ p → r	((p → q) ∧ (q → r)) → (p → r)
Disjunctive Syllogism	p ∨ q ~ p _____	((p ∨ q) ∧ (¬p)) → q

	∴ q	
Addition	p _____ ∴ p ∨ q	p → (p ∨ q)
Simplification	p ∧ q _____ ∴ p	(p ∧ q) → p p ∧ q _____ (p ∧ q) → q ∴ q
Conjunction	p q _____ ∴ p ∧ q	((p) ∧ (q)) → (p ∧ q)
Resolution	p ∨ q ¬p ∨ r _____ ∴ q ∨ r	((p ∨ q) ∧ (¬p ∨ r)) → (q ∨ r)

2.2.3 Examples:

1. **Modus Ponens:**
   Let p be "It is snowing."
   Let q be "I will study discrete math."

   "If it is snowing, then I will study discrete math." p → q
   "It is snowing." p
   "Therefore, I will study discrete math." q
2. **Modus Tollens:**
   Let p be "it is snowing."
   Let q be "I will study discrete math."
   "If it is snowing, then I will study discrete math."
   "I will not study discrete math."
   "Therefore, it is not snowing."
3. **Hypothetical Syllogism:**
   Let p be "it snows."
   Let q be "I will study discrete math."
   Let r be "I will get an A."
   "If it snows, then I will study discrete math."
   "If I study discrete math, I will get an A."
   "Therefore, If it snows, I will get an A."
4. **Disjunctive Syllogism:**
   Let p be "I will study discrete math."
   Let q be "I will study English literature."
   "I will study discrete math or I will study English literature."
   "I will not study discrete math."
   "Therefore, I will study English literature."
5. **Addition:**
   Let p be "I will study discrete math."
   Let q be "I will visit Las Vegas."

"I will study discrete math."
"Therefore, I will study discrete math or I will visit Las Vegas."

6. **Simplification:**
   Let p be "I will study discrete math."
   Let q be "I will study English literature."
   "I will study discrete math and English literature"
   "Therefore, I will study discrete math."
7. **Conjunction:**
   Let p be "I will study discrete math."
   Let q be "I will study English literature."
   "I will study discrete math."
   "I will study English literature."
   "Therefore, I will study discrete math and I will study English literature."
8. **Resolution:**
   Let p be "I will study discrete math."
   Let r be "I will study English literature."
   Let q be "I will study databases."
   "I will not study discrete math or I will study English literature."
   "I will study discrete math or I will study databases."
   "Therefore, I will study databases or I will English literature."

2.2.4 Problems:

1. **From single Proposition:**
   p ∧ (p → q)
   Show that **q** is conclusion

S No.	Step	Reason
1	p ∧ (p → q)	Premise
2	p	Simplification (1)
3	p → q	Simplification (1)
4	q	Modus Ponens (2,3)

2. **With these hypotheses:**
   "It is not sunny this afternoon and it is colder than yesterday." -p ^ q
   "We will go swimming only if it is sunny." r → p
   "If we do not go swimming, then we will take a canoe trip." -r →s
   "If we take a canoe trip, then we will be home by sunset." s →t
   ---Using the inference rules, construct a valid argument for the conclusion:
   "We will be home by sunset."

   Solution:
   1. Choose propositional variables:
   p: "It is sunny this afternoon."
   q: "It is colder than yesterday."
   r: "We will go swimming."
   s: "We will take a canoe trip."
   t: "We will be home by sunset."

2. Translation into propositional logic:

(¬ p ∧ q), (r → p), (¬ r → s), (s → t) **Conclusion:** t

S No.	Step	Reason
1	¬ p ∧ q	Premise
2	¬ p	Simplification (1)
3	r → p	Premise
4	¬ r	Modus Tollens (2,3)
5	¬ r → s	Premise
6	s	Modus Ponens (4,5)
7	s → t	Premise
8	t	Modus Ponens (6,7)

2.3 Logic:

2.3.1 What is logic?

Logic is one which consists of:

i. A formal system for describing states of affairs, consisting of a) Syntax b)Semantics.

ii. Proof Theory – a set of rules for deducing the entailment of a set sentence.

2.3.1.1 Syntax:

Syntax is the arrangement of words. Syntax of knowledge describes the possible configurations that can constitute sentences. Syntax of the language describes how to make sentences.

2.3.1.2 Semantics:

The semantics of the language defines the truth of each sentence with respect to each possible world. With this semantics, when a particular configuration exists within an agent, the agent believes the corresponding sentence.

2.3.2 First Order Logic (FOL):

Whereas propositional logic assumes the world contains facts, **first-order logic** (like natural language) assumes the world contains.

Objects: people, houses, numbers, colors, baseball games, wars, …

Relations: red, round, prime, brother of, bigger than, part of, comes between, …

Functions: father of, best friend, one more than, plus,…

2.3.2.1 FOL in BNF form: (Backus-Naur Form)

There is need to express properties of entire collections of objects instead of enumerating the objects by name. Quantifiers let us do this.

FOL contains two standard quantifiers called

- Universal (∀) and
- Existential (∃)

Formula → Primitive Formula

| (Formula Connective Formula)

| ¬ Sentence
| Quantifier Variable Formula
Primitive Formula → Predicate(Term,. . . ,Term)
Term → Function (Term,. . . ,Term)
| Constant
| Variable
Connective → ⇒
| ∧
| ∨
| ⇐⇒
Quantifier → ∀
| ∃
Constant → any string that is not used as a variable predicate or function
Variable → any string that is not used as a constant predicate or function
Predicate → any string that is not used as a constant variable or function
Function → any string that is not used as a constant variable or predicate

2.3.2.2 Universal quantification:

(∀x) P(x): means that P holds for **all** values of x in the domain associated with that variable.
E.g., (∀x) dolphin(x) => mammal(x)

2.3.2.3 Existential quantification:

(∃ x)P(x) means that P holds for **some** value of x in the domain associated with that variable
E.g., (∃ x) mammal(x) ^ lays-eggs(x)
Permits one to make a statement about some object without naming it

2.3.2.4 Examples:

Rules such as "All kings are persons," is written in first-order logic as
∀x King(x) => Person(x)
where ∀ is pronounced as " For all .."

Thus, the sentence says, "For all *x*, if *x* is a king, then z is a person."
The symbol x is called a variable (lower case letters)
The sentence ∀x P, where P is a logical expression says that P is true for every object x.

Universal quantification makes statements about every object.
It is possible to make a statement about some object in the universe without naming it,by using an existential quantifier.
Example:
"King John has a crown on his head"
∃x Crown(x) ^ OnHead (x, John)

∃x is pronounced "There exists an x such that .." or " For some x .."

1. “Not all birds can fly.”

   ¬(∀ x Bird(x) ⇒ Fly(x))

   which is the same as ∃ x Bird(x) ∧ ¬Fly(x)

2. “All birds cannot fly.”

   ∀ x Bird(x) ⇒ ¬Fly(x)

   which is the same as ¬(∃ x Bird(x) ∧ Fly(x))

3. “If anyone can solve the problem, then Hilary can.”

   (∃ x Solves(x, problem)) ⇒ Solves(Hilary, problem)

4. “Nobody in the Calculus class is smarter than everyone in the AI class”

   ¬[∃ x TakesCalculus(x) ∧ (∀ y TakesAI(y) ⇒ SmarterThan(x, y))]

5. “John hates all people who do not hate themselves.”

   ∀ x Person(x) ∧ ¬Hates(x, x) ⇒ Hates(John, x)

2.4 Chaining Methods:

After the development of the field of AI the new system called expert system came to life, and the first expert systems were formed in the 1970s, and then propagated in the 1980s. Expert systems were the first active forms of AI software. Although at the beginning there was concern about acceptance of expert system as AI program in a society, because it used a particular range of knowledge to solve narrow problems. But this idea vanished after two new concepts were proposed in 1972. Roger Schank formed the concept of "script" in 1972, and Minsky introduced the concept of "frame" in 1975. A combination of these two improved concepts gave the ability to realize the inference strategies in expert systems.

Expert system is a computer system enabling the imitation of the decision-making capability of human. They are designed to solve intricate problems by reasoning about knowledge. In general, there are three primary components in ruled-based expert system: user interface, knowledge base, and inference engine.

User interface plays the role of a bridge between the user and the expert system. This is a responsibility of a system engineer to build the user interface for a particular problem. The user interface permits the user to submit a query to the expert system in order to get an advice. So the query and the advice are considered as input and output of the expert system, respectively.

Knowledge base is very important part of the expert system. It is simply a technology that stores the rules and facts, basically in the form of **IF…THEN** statements. These statements are provided by human expert. The knowledge base demonstrates the facts that the system needs to solve the problem and to reach the goal.

Inference engine is responsible for providing the rules (logical) to the knowledge base and then generate new facts by using these rules. One of the techniques used to reach this goal is the production rules and their types: the forward chaining and the backward chaining. In addition, the inference engine needs a strong memory to save all the initial and new facts provided by the interface engine.

Rule-based systems are used in a variety of problems. For example, the CLIPS expert system was created by NASA in 1985 that led to the development the JESS expert system, which was the JAVA application of CLIPS. These two expert systems were built with the rule-based production idea. The rule-based systems are also effectıvely used in finance, law, medicine, business, manufacturing, education, computer science and computer games.

Production rules are a set of statements represented in the form of IF…THEN statements. For example, “IF A1 and B1 are true THEN C1 is true, and the action E1 must be taken“. The conditions A1 and B1 are called the left-hand side of the rule (LHS) and C1 and E1 are called the right-hand side of the rule (RHS). So if the LHS elements correspond to the facts in the working memory then the rule is activated and the RHS is executed. Another case is that the rule can be negated which means that the LHS will be only executed when no fact in the working memory matches. The actions can update the working memory by adding, modifying, or removing the facts.

The process of the output of one rule activating another rule is called chaining. Chaining technique is to break the task into small procedures and then to inform each procedure within the sequence by itself. Two types of chaining techniques are known: forward chaining and backward chaining.

2.4.1 Forward chaining:

Forward chaining is a data-driven reasoning, and starts with the known facts and tries to match the rules with these facts. There is a possibility that all the rules match the information (conditions). In forward chaining, firstly the rules looking for matching facts are tested, and then the action is executed. In the next stage the working memory is updated by new facts and the matching process all over again starts. This process is running until no more rules are left, or the goal is reached. Forward chaining is useful when a lot of information is available. Forward chaining is useful to be implemented if there are an infinite number of potential solutions like configuration problems and planning.

The term knowledge means the understanding of a specific domain. The person who has some kind of knowledge and passing it to another person is called an expert. The expert must have a wide knowledge of facts and rules, and use experience in a particular domain, and this person is called a domain expert.

Any rule-based system can be created in the presence of some components. First of all, a set of facts which are used in a working memory of a system must be available. Secondly, a set of rules must be there encompassing the possible actions to be taken. Finally, the condition must be used which is determining whether a solution is found or there is no solution at all.

It is very hard to represent the human thinking process in the form of algorithm, because this process is an internal process in the brain of a human. The expert can express

his/her knowledge as rules to describe and then solve problems. In artificial intelligence, the rules are used to represent the knowledge, and the knowledge can be better represented in the form of IF-THEN linguistic rules. In the IF part, the logical conditions are used. In THEN part some actions or results are given. The structure of a rule is easy to create and understand. For example:

IF it is raining
THEN take an umbrella

The rules are made of two parts. The IF part (left hand side or LHS) is called the condition (antecedent) and the THEN (right hand side or RHS) part is called the action (consequent).

If the conditional part of the rule (antecedent) is true, then the conclusion part (consequent) is asserted. This can be a goal or a new fact to be added to the working memory.

The rule can have more than one condition connected with (conjunction) AND, (disjunction) OR, or a mixture of both. For example:

IF < condition 1 >	IF < condition 1 >
AND < condition 2 >	OR < condition 2 >
.	
.	
AND < condition n >	OR < condition n >
THEN < action >	THEN < action >

The condition part of a rule consists of the object and its value. The object is represented in a linguistic form, and linked to its value by using an operator. Rule-based representation of knowledge for designing the production systems was proposed by Newell and Simon in the early seventies which is the foundation of the current rule-based systems. The idea came from how the human could solve a problem by implementing human knowledge represented in the form of production rules.

2.4.2 Backward Chaining:

The opposite of a forward chaining is a backward chaining, i.e. in contrast to data-driven reasoning forward chaining, a backward chaining is a goal-driven reasoning method. The backward chaining starts from the goal (from the end) which is a hypothetical solution and the inference engine tries to find the matching evidence. When it is found, the condition becomes the sub-goal, and then rules are searched to prove these sub-goals. It simply matches the RHS of the goal. This process continues until all the sub-goals are proved, and it backtracks to the previous step where a rule was chosen. If there is no rule to be established in an individual sub-goal, another rule is chosen. The backward chaining reasoning is good for the cases where there are not so much facts and the information (facts) should be generated by the user. The backward chaining reasoning is also effective for application in the diagnostic tasks.

Backward chaining works in reverse to forward chaining, and starts from the goal and tries to find data to prove its goal. Therefore, it is also called a goal-driven reasoning. After starting from the given goal, the search of THEN parts of the given rules (action part) (RHS) is conducted, and if the rule is found and its IF part (condition) matches the data in the database, then the rule is executed (fired). Otherwise, if the condition does not match

the data (facts) in the database, the inference engine sets the rule that is working on a stack and makes a new subgoal to prove the condition in the current rule. The knowledge base keeps looking for rules to prove the subgoal. The process of stacking the rules is repeated until the knowledge base has no rules to prove the subgoal.

Consider the following rules:
R1: IF A AND B THEN C
R2: IF C THEN E
R3: IF A AND E THEN H
Facts: A, B
Goal: Prove H

At first, the system looks for a rule that proves the goal, in other words, searches the RHS of the rules, which is rule 3. Then the LHS of the rule is considered which is the IF part that contains the condition of the rule. In this stage the system tries to satisfy the condition, and if the facts match the condition of this rule, then the rule fires. If not, the system makes a subgoal that contains the new rule and tries to prove it.

So in rule 3 the system looks at the LHS, and finds A and E in the database, but we only have A. Therefore, the system now will make proving E its new subgoal. E is proved in rule number 2, the LHS of rule number 2 is C. The new subgoal is proving C. The system is out the old subgoal in a stack. C is proved in rule number 1. The condition in rule number 1 can match the facts in the database. Now the system can prove the subgoals in the stack, and the goal is proved.

Backward chaining concept is used to build many types of expert systems, and especially the interactive systems simulate conversion between users and an expert person. Using backward chaining technique, the system will know what to ask and when to ask the proper question. It helps to dismantle the complex problem into easy, small, fixed sections, and the system use them automatically if needed.

Comparison between Backward Chaining and Forward Chaining:

Attribute	Backward Chaining	Forward Chaining
Also known as	Goal-driven Chaining	Data-driven Chaining
Starts from	Possible conclusion	New data
Processing	Efficient	Somewhat wasteful
Aims for	Necessary data	Any Conclusion(s)
Approach	Conservative/Cautious	Opportunistic
Practical if	Number of possible final answers is reasonable or a set of known alternatives is available	Combinatorial explosion creates an infinite number of possible right answers
Appropriate for	Diagnostic, prescription and debugging application	Planning, monitoring, control and interpretation application
Reasoning	Top-down reasoning	Bottom-up reasoning
Type of Search	Depth-first search	Breadth-first search
Who determine search	Consequents determine search	Antecedents determine search
Flow	Consequent to antecedent	Antecedent to consequent

2.5 PROCEDURAL V/S DECLARATIVE KNOWLEDGE:

Procedural knowledge	Declarative knowledge
– Knowledge about "how to do something"; e.g., to determine if Peter or Robert is older, first find their ages. – ◊ Focuses on tasks that must be performed to reach a particular objective or goal. – ◊ Examples : procedures, rules, strategies, agendas, models.	– Knowledge about "that something is true or false". e.g., A car has four tyres; Peter is older than Robert; – ◊ Refers to representations of objects and events; knowledge about facts and relationships; – ◊ Example : concepts, objects, facts, propositions, assertions, semantic nets, logic and descriptive models. –

EXAMPLES:

Types of Knowledge (4)

More Examples of Declarative and Procedural Knowledge

Procedural Knowledge	Declarative Knowledge
Process of planting herbs	Knowing something about herbs
Procedure of treating the crops of a particular disease	Description of symptoms of a plant disease
Procedure to harvest a crop	Knowledge of the month when a crop should be harvested
Draw a line graph from a data set	Familiarity with the data sets and line graphs
Procedure to register in a video lecture	Knowledge acquired in a video lecture
Type a text in a computer using a keyboard	Knowledge about the placement of keys in a keyboard

2.6 MATCHING:

To extract from the entire collection of rules those that can be applied at a given point requires some kind of **matching** between the current state and the predictions of the rules.

To do this, there are a few proposals as below:

i) Indexing
ii) Matching with variables
iii) Complex and Appropriate matching
iv) Conflict Resolution

2.6.1 INDEXING:

One way to select applicable rules is to do a simple search through all the rules, comparing each one's preconditions to the current state and extracting all the ones that match.

Instead of searching through the rules, use the current state as an **index** into the rules and select the matching ones immediately.

Eg: Consider the legal move generation rule for Chess as shown in the figure. To be able to access the appropriate rules immediately, all we need to do is assign an **index** to each board position.

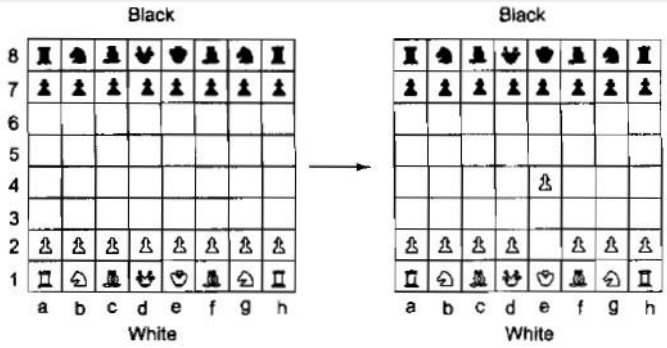


Fig. 6.4 *One Legal Chess Move*

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

→

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

Fig. 6.5 *Another Way to Describe Chess Moves*

2.6.2 Matching with Variables:

If we want to match a single condition against a single element in a state description, then, the **unification** procedure will suffice.

While it is possible to apply unification repeatedly over the cross product of preconditions and state description elements, it is more efficient to consider the many-many match problem, in which many rules are matched against many elements in the state description simultaneously.

One efficient many-many match algorithm is RETE, which gains efficiency from the three major sources:

- The temporal nature of data. RETE maintains a network of rule conditions, and it uses changes in the state description to determine which new rules might apply (and which rules might no longer apply)
- Structural similarity in rules.
  Eg: One rule concludes jaguar(x) if mammal(x), carnivorous(x) and has-spots(x).
  Another rule concludes tiger(x) if mammal(x), carnivorous(x) and has-stripes(x).

  RETE stores the rules so that they share structures in memory.

2.6.3 Complex and Approximate Matching:

A more **complex matching** process is required when the preconditions of a rule specify required properties that are not stated explicitly in the description of the current state.

Eg: A speech understanding program must contain rules that map from a description of a physical waveform to phones.

Appropriate Matching is particularly difficult to deal with because as we increase the tolerance allowed in the match, we also increase the number of rules that will match, thus increasing the size of the main search process.

Eg: Interaction with ELIZA

2.6.4 Conflict Resolution:

Sometimes, it is useful to incorporate some of that decision making into the matching process. This phase of matching process is then called Conflict Resolution.

There are three basic approaches to the problem of conflict resolution in a production system:

- Assign a preference based on the rule that matched.
- Assign a preference based on the objects that matched.
- Assign a preference based on the action that the matched rule would perform.