Knowledge and Reasoning

Unit 3

Knowledge-Based System

- A knowledge-based system is a system that uses artificial intelligence techniques to store and reason with knowledge. The knowledge is typically represented in the form of rules or facts, which can be used to draw conclusions or make decisions.
- One of the key benefits of a knowledge-based system is that it can help to automate decision-making processes. For example, a knowledge-based system could be used to diagnose a medical condition, by reasoning over a set of rules that describe the symptoms and possible causes of the condition.
- Another benefit of knowledge-based systems is that they can be used to explain their decisions to humans. This can be useful, for example, in a customer service setting, where a knowledge-based system can help a human agent understand why a particular decision was made.
- Knowledge-based systems are a type of artificial intelligence and have been used in a variety of applications including medical diagnosis, expert systems, and decision support systems.

Why use a knowledge base?

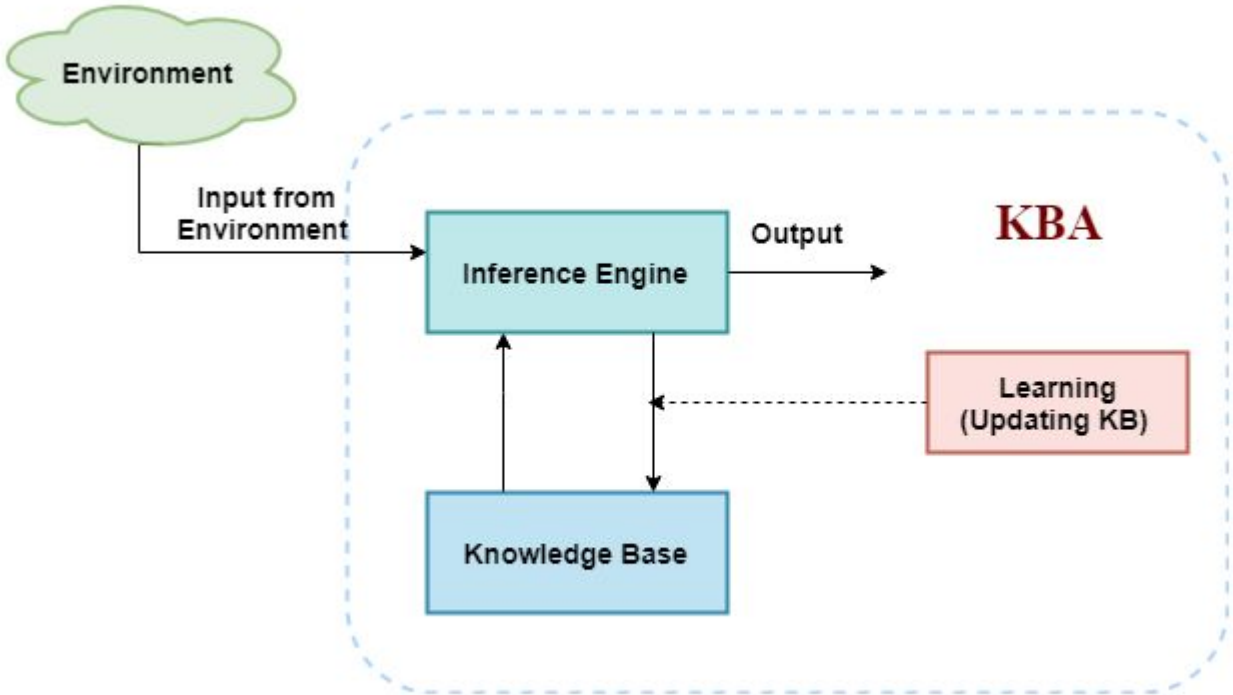
- A knowledge base **inference** is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.
- Inference means deriving new sentences from old. The inference-**based** system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. The inference system applies logical rules to the KB to deduce new information.
- **The inference** system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given:
    - Forward chaining
    - Backward chaining

- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.
- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system**.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



Operations Performed by KBA

Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

Various levels of knowledge-based agent:

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

3. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

1. Declarative approach: We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.

2. Procedural approach: In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

However, in the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

Propositional Logic in AI

Propositional logic is a fundamental part of artificial intelligence and computer science. It is a branch of mathematical logic that deals with the study of propositions, their truth values, and the logical relationships that exist between them. Propositional logic forms the basis for many AI systems, including expert systems, rule-based systems, and natural language processing.

Introduction to Propositional Logic in AI

Propositional logic is a fundamental component of artificial intelligence (AI). It is a branch of mathematical logic that deals with propositions and logical reasoning. Propositions are statements that can be true or false. In AI, propositional logic is used to represent and reason about knowledge systematically and formally. It provides a simple yet powerful way of expressing and manipulating logical relationships between propositions.

One of the key applications of propositional logic in AI is in the representation of knowledge. For example, consider the statement "All humans are mortal." This can be represented in propositional logic as a proposition, say P. Similarly, the statement "Socrates is a human" can be represented as another proposition, say Q. The logical relationship between P and Q can be represented using logical connectives, such as "and," "or," and "not." For instance, the proposition "All humans are mortal, and Socrates is a human" can be represented as $P \wedge Q$.

Propositional logic is also used in AI to reason about the relationships between propositions. Given a set of propositions, we can use logical inference rules to derive new propositions. For example, if we know that $P \wedge Q$ is true, and we also know that $P \rightarrow R$, we can infer that $Q \rightarrow R$ is also true.

Some Basic Facts About Propositional Logic

Here are some basic facts about propositional logic in AI:

- Propositional logic is a formal language that uses symbols to represent propositions and logical connectives to combine them. The symbols used in propositional logic include letters such as p, q, and r, which represent propositions, and logical connectives such as $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation), which are used to combine propositions.
- An statement is a proposition if it is either true or false. Examples of propositions include "2+2=4," and "The sky is blue."
- Logical connectives are used to combine propositions to form more complex statements.
- Truth tables are used to represent the truth values of propositions and the logical connectives that combine them.
- In propositional logic, inference rules are used to derive new propositions from existing ones.
- Propositional logic is a limited form of logic that only deals with propositions that are either true or false.

Syntax of Propositional Logic

Syntax of propositional logic refers to the formal rules for constructing statements in propositional logic. Propositional logic deals with the study of propositions, which are declarative statements that are either true or false. The syntax of propositional logic consists of two main components: atomic propositions and compound propositions.

Atomic Propositions

Atomic propositions are simple statements that cannot be broken down into simpler statements. They are the building blocks of propositional logic. An atomic proposition can be represented by a letter or symbol, such as p, q, r, or s. For example, the following are atomic propositions:

p: The sky is blue. q: The grass is green. r: 2+2=4. s: The Earth orbits the Sun.

Compound Propositions

Compound propositions are formed by combining atomic propositions using logical operators. There are several logical operators in propositional logic, including negation, conjunction, disjunction, implication, and equivalence.

Example:

- "It is raining today, and state it is wet."
- "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives

When connecting two simpler assertions or logically expressing a statement, logical connectives are used. Using logical connectives, we can build compound propositions. The following list of connectives includes the main five:

Negation

The negation of a proposition p is denoted by ¬p and is read as "not p". For example: ¬p: The sky is not blue.

Conjunction

The conjunction of two propositions p and q is denoted by p ∧ q and is read as "p and q's". The conjunction is true only if both p and q are true. For example: p ∧ q`: The sky is blue and the grass is green.

Disjunction

The disjunction of two propositions p and q is denoted by p ∨ q and is read as "p or q". The disjunction is true if at least one of p and q is true. For example: p ∨ q: The sky is blue or the grass is green.

Implication

The implication of two propositions p and q is denoted by p → q and is read as "if p then q". The implication is false only if p is true

Biconditional

A sentence such as P⇔ Q is a Biconditional sentence, example I will eat lunch if and only if my mood improves. P= I will eat lunch, Q= if my mood improves, it can be represented as P ⇔ Q.

Summarized table for Propositional Logic Connectives

Connective	Name	Meaning	Example
$\neg$	Negation	"Not"	$\neg p$ ("not p") means "it is not the case that p"
$\wedge$	Conjunction	"And"	$p \wedge q$ ("p and q") means "both p and q are true"
$\vee$	Disjunction	"Or"	$p \vee q$ ("p or q") means "either p or q is true (or both)"
$\oplus$	Exclusive Disjunction	"Exclusive Or"	$p \oplus q$ ("p xor q") means "either p or q is true, but not both"
$\rightarrow$	Implication	"If...then"	$p \rightarrow q$ ("if p then q") means "if p is true, then q must be true"
$\leftrightarrow$	Bi-implication	"If and only if"	$p \leftrightarrow q$ ("p iff q") means "p is true if and only if q is true"

Truth table with Three Propositions

In propositional logic, a truth table is a tool used to calculate the true value of a compound proposition based on the truth values of each of its constituent propositions.

Let's consider a truth table with three propositions: p, q, and r.

p	q	r	p AND q	(p AND q) OR r
T	T	T	T	T
T	T	F	T	T
T	F	T	F	T
T	F	F	F	F
F	T	T	F	T
F	T	F	F	F
F	F	T	F	T
F	F	F	F	F

Precedence of Connectives

In propositional logic, there are several types of connectives, each with its own precedence rules. Precedence of connectives refers to the order in which the logical connectives are evaluated when there are multiple connectives in a logical expression.

These connectives have different levels of precedence, and the order of evaluation is as follows:

- Negation (~) - The negation connective has the highest precedence and is evaluated first. It takes a single proposition and negates it, changing its truth value.
- Conjunction (&) - The conjunction connective has the second-highest precedence and is evaluated next. It takes two propositions and evaluates to true only if both propositions are true.
- Disjunction (|) - The disjunction connective has the third-highest precedence and is evaluated next. It takes two propositions and evaluates them to be true if at least one of the propositions is true.
- Conditional (->) - The conditional connective has the fourth-highest precedence and is evaluated next. It takes two propositions and evaluates to false only if the antecedent is true and the consequent is false; otherwise, it evaluates to true.
- Biconditional (<->) - The biconditional connective has the lowest precedence and is evaluated last. It takes two propositions and evaluates to true only if both propositions have the same truth value.

Logical Equivalence

Logical equivalence is an important concept in propositional logic. In propositional logic, logical equivalence is a relationship between two propositional formulas, which means that the two formulas have the same truth value in all possible cases. This relationship is denoted by the symbol "≡" or "off" (if and only if).

In other words, two propositional formulas are logically equivalent if they are always true or always false together. For example, the propositional formulas "P ∧ Q" and "Q ∧ P" are logically equivalent because they have the same truth value for all possible values of P and Q. Similarly, "¬(P ∨ Q)" and "(¬P) ∧ (¬Q)" are logically equivalent.

Logical equivalence is important in propositional logic because it allows us to simplify complex formulas and reason about them more effectively.

Properties of Operators

In propositional logic, operators are used to combine propositions to form more complex propositions`. Here are some of the properties of operators commonly used in propositional logic:

Set 1: A = {2, 4, 6} B = {3, 6, 9}

- Commutativity: The commutative property states that the order in which the operators are applied does not affect the result. For example, the union of A and B is A ∪ B = {2, 3, 4, 6, 9}, which is the same as the union of B and A, i.e., B ∪ A = {2, 3, 4, 6, 9}. Therefore, the union operator is commutative.
- Associativity: The associative property states that the grouping of operators does not affect the result. For example, the intersection of A, B, and C is (A ∩ B) ∩ C = {6} ∩ {3, 6, 9} = {6} and A ∩ (B ∩ C) = {2, 4, 6} ∩ {3, 6, 9} = {6}. Therefore, the intersection operator is associative.
- Identity element: The identity element property states that there is an element that can be combined with any other element using the operator without changing the result. For example, the intersection of A and the universal set U is A ∩ U = {2, 4, 6}. Therefore, the universal set is the identity element of the intersection operator.
- Distributive: The distributive property states that one operator can be distributed over the other operator. For example, the intersection operator is distributive over the union operator. That is, A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C). For instance, A ∩ (B ∪ C) = {2, 4, 6} ∩ {3, 6, 7} = {6}, and (A ∩ B) ∪ (A ∩ C) = {2, 4, 6} ∩ {6} ∪ {2, 4, 6} ∩ {3, 7} = {6}.
- De Morgan's Law: De Morgan's Law states that the complement of the union of two sets A and B is equal to the intersection of the complement of A and the complement of B. That is, (A ∪ B)' = A' ∩ B'. For example, (A ∪ B)' = {2, 3, 4, 6, 9}' = {1, 5, 7, 8} and A' ∩ B' = {1, 3, 5, 7, 8} ∩ {1, 2, 4, 5, 7, 8} = {1, 5, 7, 8}. Therefore, De Morgan's Law holds.
- Double-negation elimination: Double negation elimination is a valid rule of replacement that states that if not not-A is true, then A is true. For example, if A = {2, 4, 6}, then A'' = {2, 4, 6}, which is equivalent to A. Therefore, double negation elimination holds.

Limitations of Propositional Logic

Some of the most significant limitations are:

- Limited expressivity: Propositional logic is limited in its ability to represent complex relationships between objects or concepts. It can only express simple propositional statements with binary truth values (true/false). This makes it difficult to represent concepts such as uncertainty, ambiguity, and vagueness.
- Inability to handle quantifiers: Propositional logic is unable to handle quantifiers such as "all" or "some." For example, it cannot represent the statement "all humans are mortal" in a concise manner. This makes it difficult to reason about sets of objects or concepts.
- Lack of support for negation: Propositional logic does not provide an easy way to represent negation. This can make it difficult to represent negative statements and reason about them.
- Difficulty with recursive structures: Propositional logic struggles to represent recursive structures, such as lists or trees, which are common in many AI applications. Recursive structures require a more expressive language, such as first-order logic or higher-order logic.
- Inability to handle temporal relationships: Propositional logic is not well-suited for representing temporal relationships between events or states. It cannot represent concepts such as causality or temporal precedence, which are crucial in many AI applications.

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:
    a. **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
    b. **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
    c. **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
    a. **Syntax**
    b. **Semantics**

Syntax of First-Order logic:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

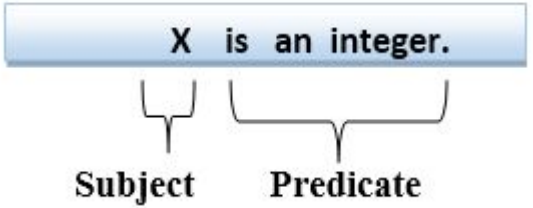
Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

Example: Ravi and Ajay are brothers: ay).
Chinky is a cat: => cat (Chinky).



Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.

Quantifiers in First-order logic:

- 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.

  There are two types of quantifier:

  a. **Universal Quantifier, (for all, everyone, everything)**

  b. **Existential quantifier, (for some, at least one).**

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.

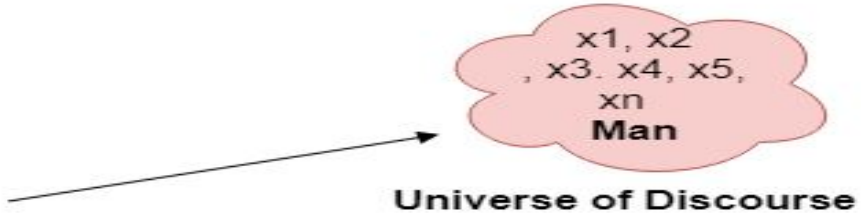
- **For all x**
- **For each x**
- **For every x.**

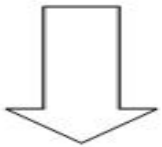
Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:

- **x1 drinks coffee**
  ^
- **x2 drinks**
  ^
- **x3 drinks milk**
  ^
- .
- .
  ^
- **xn drinks milk**





So in shorthand notation, we can write it as :

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

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.

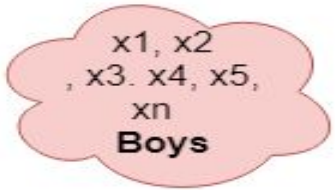
If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.

- **x1 is intelligent**
  V
- **x2 is intelligentV**
- **x3 is intelligent**
  V
- .
- .
  V
- **xn is intelligent**



Universe of Discourse

So in short-hand notation, we can write it as:

∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier $\forall$ is implication $\rightarrow$.
- The main connective for existential quantifier $\exists$ is and $\wedge$.

Properties of Quantifiers:

- In universal quantifier, $\forall x \forall y$ is similar to $\forall y \forall x$.
- In Existential quantifier, $\exists x \exists y$ is similar to $\exists y \exists x$.
- $\exists x \forall y$ is not similar to $\forall y \exists x$.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.
∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y),**" **where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:
∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃ , and it will be represented as**:
∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y),**" **where x= student, and y= subject**.
Since there are not all students, so we will use **∀ with negation, so** following representation for this:
¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

Forward chaining and backward chaining are two strategies used in designing expert systems for artificial intelligence. Forward chaining is a form of reasoning that starts with simple facts in the knowledge base and applies inference rules in the forward direction to extract more data until a goal is reached. Backward chaining starts with the goal and works backward, chaining through rules to find known facts that support the goal.

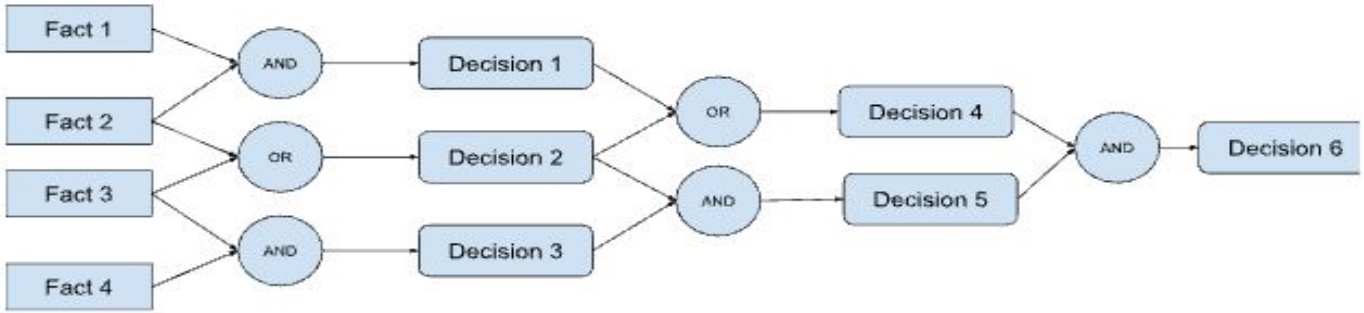
They influence the type of expert system you'll build for your AI. An expert system is a computer application that uses rules, approaches and facts to provide solutions to complex problems.

- **Forward chaining:** Forward chaining is a form of reasoning for an AI expert system that starts with simple facts and applies inference rules to extract more data until the goal is reached.
- **Backward chaining:** Backward chaining is another strategy used to shape an AI expert system that starts with the end goal and works backward through the AI's rules to find facts that support the goal.

What Is Forward Chaining?

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. The forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied and adds their conclusion to the known facts. This process repeats until the problem is solved.

In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information. An endpoint, or goal, is achieved through the manipulation of knowledge that exists in the knowledge base.



Forward Chaining Properties

- Forward chaining follows a down-up strategy, going from bottom to top.
- It uses known facts to start from the initial state (facts) and works toward the goal state, or conclusion.
- The forward chaining method is also known as data-driven because we achieve our objective by employing available data.
- The forward chaining method is widely used in expert systems such as CLIPS, business rule systems and manufacturing rule systems.
- It uses a breadth-first search as it has to go through all the facts first.
- It can be used to draw multiple conclusions.

Examples of Forward Chaining

Let's say we want to determine the max loan eligibility for a user and cost of borrowing based on a user's profile and a set of rules, both of which constitute the knowledge base. This inquiry would form the foundation for our problem statement.

KNOWLEDGE BASE

Our knowledge base contains the combination of rules and facts about the user profile.

1. John's credit score is 780.
2. A person with a credit score greater than 700 has never defaulted on their loan.
3. John has an annual income of $100,000.
4. A person with a credit score greater than 750 is a low-risk borrower.
5. A person with a credit score between 600 to 750 is a medium-risk borrower.
6. A person with a credit score less than 600 is a high-risk borrower.
7. A low-risk borrower can be given a loan amount up to 4X of his annual income at a 10 percent interest rate.
8. A medium-risk borrower can be given a loan amount of up to 3X of his annual income at a 12 percent interest rate.
9. A high-risk borrower can be given a loan amount of up to 1X of his annual income at a 16 percent interest rate.

QUESTION

Next, we'll seek to find answers to two questions:

1. What max loan amount can be sanctioned for John?
2. What will the interest rate be?

RESULTS

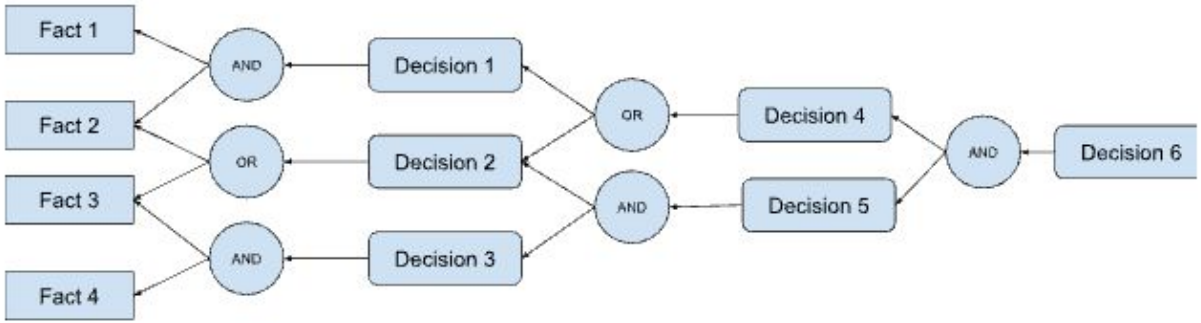
To deduce the conclusion, we apply forward chaining on the knowledge base. We start from the facts which are given in the knowledge base and go through each one of them to deduce intermediate conclusions until we are able to reach the final conclusion or have sufficient evidence to negate the same.

```
John' CS = 780   AND  CS > 750 are Low Risk Borrower → John is a Low Risk Borrower

Loan Amount for Low Risk Borrower is 4X annual income AND John's annual income is $100k

→ Max loan amount that can be sanctioned is $400k at a 10% interest rate.
```

What Is Backward Chaining

Backward chaining is also known as a backward deduction or backward reasoning method when using an inference engine. In this, the inference engine knows the final decision or goal. The system starts from the goal and works backward to determine what facts must be asserted so that the goal can be achieved.

For example, it starts directly with the conclusion (hypothesis) and validates it by backtracking through a sequence of facts. Backward chaining can be used in debugging, diagnostics and prescription applications.



Properties of Backward Chaining

- Backward chaining uses an up-down strategy going from top to bottom.
- The modus ponens inference rule is used as the basis for the backward chaining process. This rule states that if both the conditional statement `(p->q)` and the antecedent `(p)` are true, then we can infer the subsequent `(q)`.
- In backward chaining, the goal is broken into sub-goals to prove the facts are true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- The backward chaining algorithm is used in game theory, automated theorem-proving tools, inference engines, proof assistants and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

Examples of Backward Chaining

In this example, let's say we want to prove that John is the tallest boy in his class. This forms our problem statement.

KNOWLEDGE BASE

We have few facts and rules that constitute our knowledge base:

- John is taller than Kim
- John is a boy
- Kim is a girl
- John and Kim study in the same class
- Everyone else other than John in the class is shorter than Kim

QUESTION

We'll seek to answer the question:

- Is John the tallest boy in class?

RESULTS

Now, to apply backward chaining, we start from the goal and assume that John is the tallest boy in class. From there, we go backward through the knowledge base comparing that assumption to each known fact to determine whether it is true that John is the tallest boy in class or not.

Our goal:

- John is the tallest boy in the class

Height (John) > Height (anyone in the class)

AND

John and Kim both are in the same class

AND

Height (Kim) > Height (anyone in the class except John)

AND

John is boy

SO

Height (John) > Hight(Kim)

Which aligns with the knowledge base fact. Hence the goal is proved true.

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \ldots\ldots \vee l_k, \qquad m_1 \vee \ldots\ldots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

Where $\mathbf{l_i}$ and $\mathbf{m_j}$ are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] **and** **[¬ Loves(a, b) V ¬ Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)] **and** **[¬ Loves(a, b) V ¬Kills(a, b)]**

Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**

These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Thank you