USN | | | | | | | | | | |

Sub:	AI					Sub Code:	BAD402
Date:	2.5.25	Duration:	90 minutes	Max Marks:	50	Sem/Sec:4 A,B,C	

Scheme and Solutions

The N-Queens problem involves placing N queens on an N x N chessboard such that no two queens threaten each other. This means that no two queens can be in the same row, column, or diagonal. You are required to solve this problem using the Hill Climbing search strategy.

Answer:- (5+5M)

Hill-climbing is a greedy local search algorithm. It continuously moves in the direction of increasing value (or decreasing cost) and stops when it reaches a "peak" (local maximum or plateau) where no neighboring state is better.

This algorithm:

- Doesn't look ahead beyond the immediate neighbors.
- Doesn't maintain a full search tree or path history.
- Can get stuck on local maxima, plateaux, or ridges.

1 a State Space Landscape Concepts

These concepts are analogies used to describe how solutions are distributed in the search space:

- Local Maximum: A state better than its neighbors, but not the best overall.
- Plateau: A flat region with many states of the same value.
    - Includes shoulders (edges from which progress is possible).
- Ridges: Narrow paths of increasing values that are hard to navigate.

The “porcupine” analogy humorously describes highly rugged landscapes with many tiny local maxima (i.e., very difficult search spaces).

Variants of Hill-Climbing

1. Stochastic Hill Climbing:
    - Randomly chooses among uphill moves.
    - May escape local maxima better than basic hill climbing.
2. First-Choice Hill Climbing:

- Evaluates successors in random order and picks the first better one.

- Useful when the branching factor (number of successors) is high.

3. Random-Restart Hill Climbing:
    - Repeatedly restarts from random states until a goal is found.
    - Complete with high probability because it can eventually reach any part of the state space.
    - Example: In the 8-queens problem, it finds solutions quickly with few restarts.

Allowing Sideways Moves

- Sideways Move: Moving to a neighbor of equal value to escape plateaux.
- Risk: Could enter an infinite loop if not handled properly.
- Solution: Limit the number of allowed sideways moves (e.g., 100).
- Result: Increases success rate (e.g., from 14% to 94% in the 8-queens problem).
- Cost: More steps per success/failure.

Success and Limitations

- Strengths:
    - Simple and fast.
    - Works well when the search space has few local maxima.
- Weaknesses:
    - Gets stuck in local maxima or flat regions.
    - No backtracking or memory of past states.
- Improvements:
    - Using random restarts or sideways moves.
    - Applying variations like stochastic or first-choice strategies.

N-Queens Problem: Hill-Climbing Example

Here n=8. Place 8 queens on an 8×8 chessboard such that no two queens attack each other — i.e., no two queens share the same row, column, or diagonal.State Space Representation

- Each state is an arrangement of 8 queens, one per column.
- The row position of each queen in its column is the variable to change.
- This reduces the state space significantly (only 8 positions per queen in 8 columns).

So, a state can be represented as a list of 8 numbers, e.g., [0, 4, 7, 5, 2, 6, 1, 3], where the index is the column and the value is the row of the queen.

Heuristic Function (Cost Function)

- h(state) = Number of pairs of queens attacking each other.
- Goal is to minimize h. Perfect solutions have h = 0.

Example:

- If 3 pairs of queens are attacking each other, h = 3.

Hill-Climbing Approach

1. Start with a random state (queens placed randomly).
2. Evaluate all neighboring states (by moving one queen in its column to a different row).
3. Choose the neighbor with the lowest h value.
4. Move to that neighbor.
5. Repeat until:
    - No neighbor has a better (lower) h → local minimum, or
    - h = 0 → goal reached.

Limitations Observed

- Local maxima: States where all neighbors are worse, but it's not the best (i.e., $h \neq 0$).
- Plateaux: Regions where many neighboring states have the same h. Hard to know which direction to go.
- Ridges: Sequences of moves that must be followed carefully, but are hard to find with local-only search.

Variants and Solutions for 8-Queens

Random-Restart Hill Climbing

- Repeats the hill-climbing algorithm from random initial states until a solution is found.
- Highly effective:
    - Solves the problem quickly and reliably.
    - Works even for larger n-queens problems (e.g., 1 million queens in under a minute).

Sideways Moves

- Allow moves to neighboring states with the same h value.
- Helps escape plateaux.

- Limiting to a max number (e.g., 100) prevents infinite loops.
- Success rate jumps:
    - From 14% (no sideways move) → 94% (with sideways move in 8-queens).

Example: Search Performance

- On average, 4 steps if successful, 3 steps if stuck.
- Search space has 88≈17 million states8^8 \approx 17 \text{ million states}88≈17 million states, but hill climbing doesn't explore all.
- Random-restart hill climbing finds solutions in seconds even for complex versions.

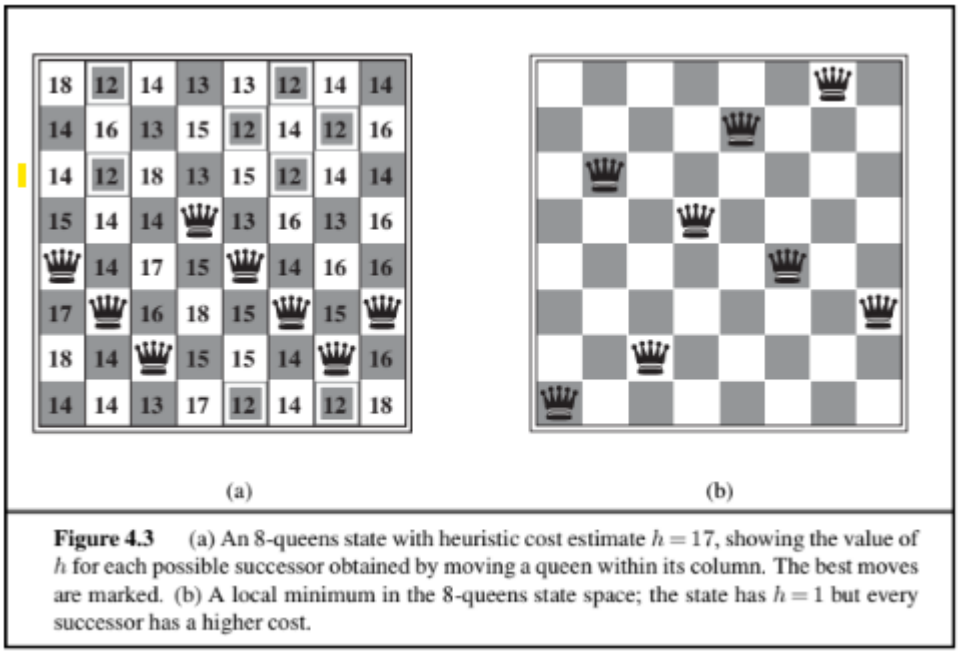


Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

2	a	Explain syntax and semantics for first order logic **Answer: - (2.5+2.5 M)** FOL has sentences, and also has terms, which signify objects. Variables, functions and Constant symbols are utilized to create terms, predicate symbols and quantifiers are utilized to create sentences. **Models for FOL** The domain of the representation is a group of objects it have; these objects are occasionally known as domain elements. Figure: displays a pattern with 5 objects.

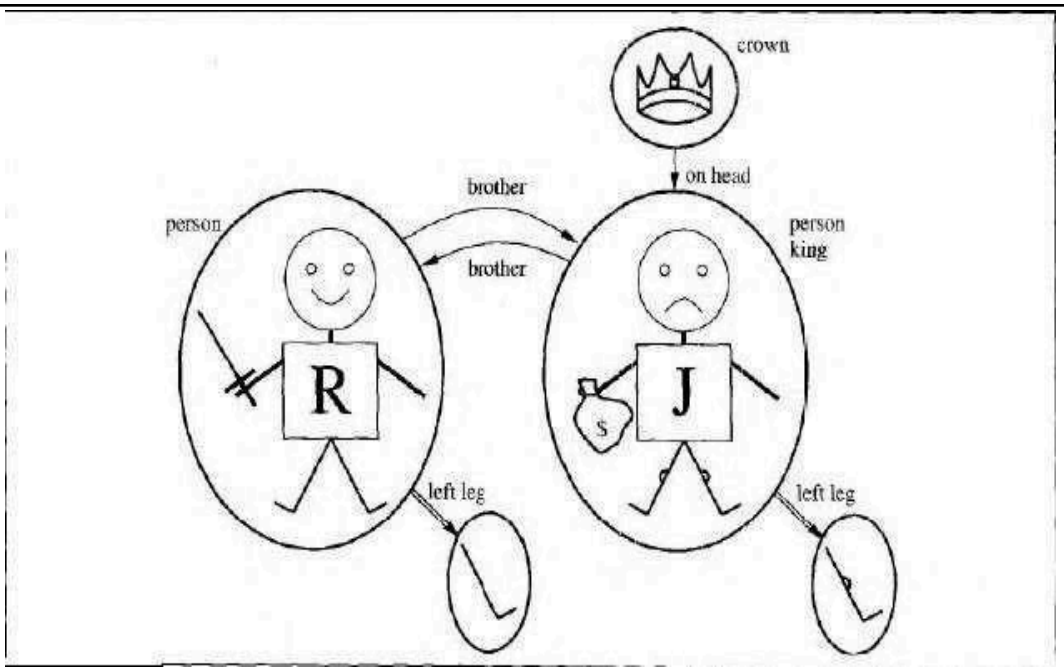


Figure: A Model Having 5 Objects Three Unary Relations, Two Binary Relations and One Unary Function

Symbols and Interpretations

Figure shows the formal grammar of FOL**.**

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable,\ldots\ Sentence \\
&\mid \neg\ Sentence \\
\\
AtomicSentence &\rightarrow Predicate(Term,\ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term,\ldots) \\
&\mid Constant \\
&\mid Variable \\
\\
Connective &\rightarrow \Rightarrow \mid \wedge \mid V \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid 3 \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLeg \mid \ldots
\end{aligned}
$$

Figure: The Syntax of FOL with Equality, Specified in BNF

The basic semantic components of the FOL are the signs that apply for functions, relations and objects. The signs arrive in 3 types: **function symbols** apply for functions; **constant symbols** apply for objects; and **predicate symbols** apply for relations.

Terms are a logical statement, which indicates to an object.

An **atomic statement** is created from a predicate symbol pursued by a parenthesized listing of expressions.

The **atomic statement** is correct in a specified model, beneath a specified clarification, if the relationship indicated by the predicate symbol keeps amongst the objects indicated by the arguments.

Quantifiers: Theses are used to state properties of total collection of objects, than presenting the objects by the name.

First order logic (FOL) contains 2 quantifiers:

1. Existential quantifiers
2. Universal quantifiers

Equality

FOL consists of one additional way to build atomic sentences, excluding a terms and predicate as described in advance. We may apply the **equality sign** to create sentences to work that 2 expressions indicate to the similar object.

b

Explain unification and lifting in detail.

Answer: - (2.5+2.5 M)

Lifting and Unification

A First Order Inference Rule

This inference procedure may be confined as a distinct inference rule, which we term General.

GENERALIZEPONEDN S **sized Modus-Ponens:** The atomic statement q and *p,,,, p,',*

Wherever there is a replacement *eight* such *Suesr(Q,pil)=Suesr(B,p,),* for all *i,*

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Unification

Lifted-inference rules need discovering replacements, which create dissimilar reasonable statements.

UNIFICATION looks the same. This procedure is known as **unification** and it is a crucial element of every first-order UNIFIER inference algorithms.

The ***UNIFY* algorithm** obtain 2 statements and give back a **unifier** for them if one be present:

UNIFY(^,q)=**0** wherever *SUBST(O,= ~)SUBST(@ q,).*

The problem happens because the 2 sentences occur to utilize the similar name of a variable. The problem may be keep away by **regularizing separately** one of the 2 statements is combined that means changing the name of its APART variables to keep away name conflicts.

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
          y, a variable, constant, list, or compound
          θ, the substitution built up so far (optional, defaults to empty)

  if θ = failure then return failure
  else if x = y then return θ
  else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) and COMPOUND?(y) then
      return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) and LIST?(y) then
      return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure
```

```
function UNIFY-VAR(var, x, θ) returns a substitution
  inputs: var, a variable
          x, any expression
          θ, the substitution built up so far

  if {var/val} ∈ θ then return UNIFY(val, x, θ)
  else if {x/val} ∈ θ then return UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) then return failure
  else return add {var/x} to θ
```

Figure: The UNIFY Algorithm

3 a

Write and explain forward chaining algorithm with an example

Answer:- (7+3M)

The forward-chaining algorithm PL-FC-ENTAILS?(KB, q) determines if a single proposition symbol q—the query—is entailed by a knowledge base of definite clauses. It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts. For example, if L1,1 and Breeze are known and (L1,1 ∧ Breeze) ⇒ B1,1 is in the knowledge base, then B1,1 can be added. This process continues until the query q is added or until no further inferences can be made. The detailed algorithm is shown in Figure 7.15; the main point to remember is that it runs in linear time. The best way to understand the algorithm is through an example and a picture. Figure 7.16(a) shows a simple knowledge base of Horn clauses with A and B as known facts. Figure 7.16(b) shows the same knowledge base drawn as an AND–OR graph (see Chapter 4). In AND–OR graphs, multiple links joined by an arc indicate a conjunction—every link must be proved—while multiple links without an arc indicate a disjunction—any link can be proved. It is easy to see how forward chaining works in the graph. The known leaves (here, A and B) are set, and inference propagates up the graph as far as possible. Wherever a conjunction appears, the propagation waits until all the conjuncts are known before proceeding. The reader is encouraged to work through the example in

detail. It is easy to see that forward chaining is sound: every inference is essentially an application of Modus Ponens. Forward chaining is also complete: every entailed atomic sentence
will be derived. The easiest way to see this is to consider the final state of the inferred table (after the algorithm reaches a fixed point where no new inferences are possible).

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

Figure 7.15 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

The table contains true for each symbol inferred during the process, and false for all other symbols. We can view the table as a logical model; moreover, every definite clause in the original KB is true in this model. To see this, assume the opposite, namely that some clause a1 $\wedge$ ... $\wedge$ ak $\Rightarrow$ b is false in the model. Then a1 $\wedge$ ... $\wedge$ ak must be true in the model and b must be false in the model. But this contradicts our assumption that the algorithm has reached a fixed point! We can conclude, therefore, that the set of atomic sentences inferred at the fixed point defines a model of the original KB. Furthermore, any atomic sentence q that is entailed by the KB must be true in all its models and in this model in particular. Hence, every entailed atomic sentence q must be inferred by the algorithm. Forward chaining is an example of the general concept of data-driven reasoning—that is, reasoning in which the focus of attention starts with the known data. It can be used within an agent to derive conclusions from incoming percepts, often without a specific query in mind. For example, the wumpus agent might TELL its percepts to the knowledge base using an incremental forward-chaining algorithm in which new facts can be added to the agenda to initiate new inferences. In humans, a certain amount of data-driven reasoning occurs as new information arrives. For example, if I am indoors and hear rain starting to fall, it might occur to me that the picnic will be canceled. Yet it will

probably not occur to me that the seventeenth petal on the largest rose in my neighbor's garden will get wet; humans keep forward chaining under careful control, lest they be swamped with irrelevant consequences.

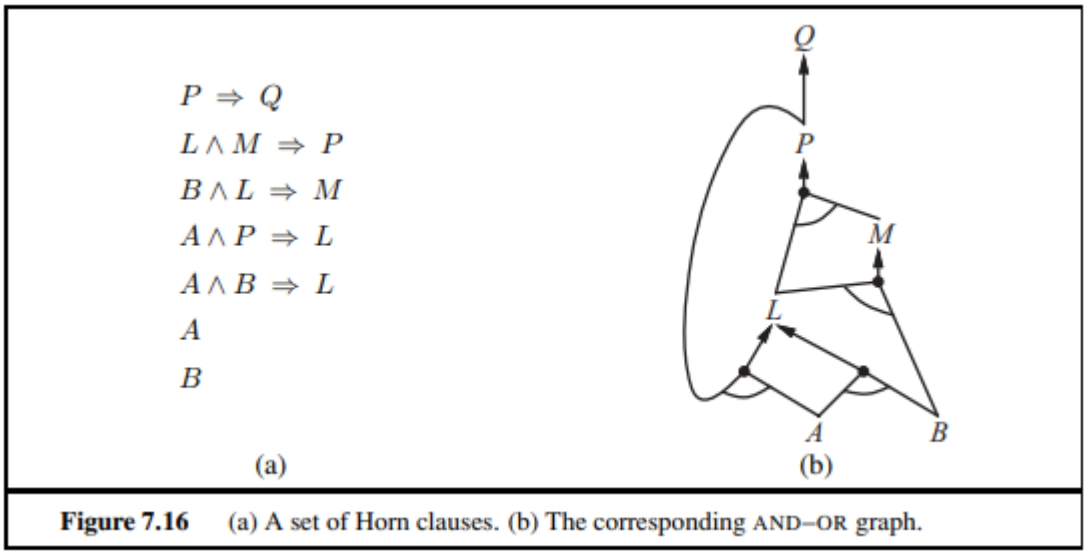


Figure 7.16 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

4

Explain basic probability notation in detail.

Answer:- (10M)

What probabilities are about

Like logical assertions, probabilistic assertions are about possible worlds. Whereas logical assertions say which possible worlds are strictly ruled out (all those in which the assertion is false), probabilistic assertions talk about how probable the various worlds are. In probability theory, the set of all possible worlds is called the **sample space**. The possible worlds are mutually exclusive and exhaustive—two possible worlds cannot both be the case, and one possible world must be the case. For example, if we are about to roll two (distinguishable) dice, there are 36 possible worlds to consider: (1,1), (1,2), ..., (6,6). The Greek letter Ω (uppercase omega) is used to refer to the sample space, and ω (lowercase omega) refers to elements of the space, that is, particular possible worlds.

A fully specified probability model associates a numerical probability P (ω) with each possible world.1 The basic axioms of probability theory say that every possible world has a probability between 0 and 1 and that the total probability of the set of possible worlds is 1:

- Sample space Ω: the set of all possible worlds
- $\omega \in \Omega$ is a possible world (aka sample point or atomic event) ex: the dice roll (1,4)
- the possible worlds are mutually exclusive and exhaustive. ex: the 36 possible outcomes of rolling two dice: (1,1), (1,2), ...
- A probability model (aka probability space) is a sample space with an assignment $P(\omega)$ for every $\omega \in \Omega$ s.t.
  - $0 \leq P(\omega) \leq 1$, for every $\omega \in \Omega$ $\Sigma\omega \in \Omega P(\omega) = 1$
- Ex: 1-die roll: $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$
- An Event A is any subset of Ω, s.t. $P(A) = \Sigma\omega \in A P(\omega)$
  - events can be described by propositions in some formal language ex: $P(Total = 11) = P(5, 6) + P(6, 5) = 1/36 + 1/36 = 1/18$
  - ex: $P(doubles) = P(1, 1) + P(2, 2) + ... + P(6, 6) = 6/36 = 1/6$

Probabilities such as P (Total = 11) and P (doubles) are called **unconditional or prior probabilities** (and sometimes just "priors" for short); they refer to degrees of belief in propositions in the absence of any other information.

Unconditional or prior probabilities refer to degrees of belief in propositions in the absence of any other information (evidence)

ex: $P(cavity) = 0.2$, $P(Total = 11) = 1/18$, $P(double) = 1/6$

Conditional or posterior probabilities refer to degrees of belief in proposition a given some evidence b: P(a|b)

- evidence: information already revealed
- ex: $P(cavity \mid toothache) = 0.6$: p. of a cavity given a toothache (assuming no other information is provided!)
- ex: $P(Total = 11 \mid die1 = 5) = 1/6$: p. of total 11 given first die is 5
- restricts the set of possible worlds to those where the first die is 5

Note: $P(a \mid ... \wedge a) = 1$, $P(a \mid ... \wedge \neg a) = 0$. ex: $P(cavity \mid toothache \wedge cavity) = 1$, $P(cavity \mid toothache \wedge \neg cavity) = 0$

- Less specific belief still valid after more evidence arrives. ex: $P(cavity) = 0.2$ holds even if $P(cavity \mid toothache) = 0.6$

- New evidence may be irrelevant, allowing for simplification. ex: $P(cavity \mid toothache, 49ersWin) = P(cavity \mid toothache) = 0.8$

Conditional probability: $P(a|b) \stackrel{def}{=} \frac{P(a \wedge b)}{P(b)}$, s.t. $P(b) > 0$

ex: $P(Total = 11 | die_1 = 5) = \frac{P(Total = 11 \wedge die_1 = 5)}{P(die_1 = 5)} = \frac{1/6 \cdot 1/6}{1/6} = 1/6$

observing b restricts the possible worlds to those where b is true

Production rule: $P(a \wedge b) = P(a|b) \cdot P(b) = P(b|a) \cdot P(a)$

Production rule for whole distributions: $\mathbf{P}(X, Y) = \mathbf{P}(X|Y) \cdot \mathbf{P}(Y)$

ex: $\mathbf{P}(Weather, Cavity) = \mathbf{P}(Weather|Cavity)\mathbf{P}(Cavity)$, that is:

$P(sunny, cavity) = P(sunny|cavity)P(cavity)$

...

$P(snow, \neg cavity) = P(snow|\neg cavity)P(\neg cavity)$

a 4×2 set of equations, not matrix multiplication!

Chain rule is derived by successive application of product rule:

$\mathbf{P}(X_1, ..., X_n)$
$= \mathbf{P}(X_1, ..., X_{n-1})\mathbf{P}(X_n|X_1, ..., X_{n-1})$
$= \mathbf{P}(X_1, ..., X_{n-2})\mathbf{P}(X_{n-1}|X_1, ..., X_{n-2})\mathbf{P}(X_n|X_1, ..., X_{n-1})$
$= ...$
$= \prod_{i=1}^{n} \mathbf{P}(X_i|X_1, ..., X_{i-1})$

Th

- Factored representation of possible worlds: sets of ⟨*variable, value*⟩ pairs Variables in probability theory: **Random variables**
- **domain:** the set of possible values a variable can take on
- ex: Die: {1, 2, 3, 4, 5, 6}, Weather: {*sunny, rain, cloudy, snow* }, Odd: {*true, false*}
- A random variable can be seen as a function from sample points to the domain: ex: *Die*(ω), *Weather* (ω),... ("(ω)" typically omitted)
- Probability Distribution gives the probabilities of all the possible values of a random variable

$X: P(X = x_i) \stackrel{def}{=} \sum_{\omega \in X(\omega)} P(\omega)$

ex: $P(Odd = true) = P(1) + P(3) + P(5) = 1/6 + 1/6 + 1/6 = 1/2$

"The probability that the patient has a cavity, given that she is a teenager with no toothache, is 0.1" as follows:

P (cavity |¬toothache A teen)=0.1 .

Probability Distribution gives the probabilities of all the possible values of a random variable
ex: P (Weather = sunny)=0.6 P
(Weather = rain)=0.1
P (Weather = cloudy)=0.29 P
(Weather = snow)=0.01 ,
but as an abbreviation we will allow
P(Weather)= 0.6, 0.1, 0.29, 0.01

where the bold P indicates that the result is a vector of numbers, and where we assume a pre defined ordering sunny, rain , cloudy , snow on the domain of Weather. We say that the P statement defines a probability distribution for the random variable Weather.TheP notation is also used for conditional distributions: P(X | Y) gives the values of P (X = xi | Y = yj) for each possible i, j pair.

For continuous variables, it is not possible to write out the entire distribution as a vector, because there are infinitely many values. Instead, we can define the probability that a random variable takes on some value x as a parameterized function of x. For example, the sentence P (NoonTemp = x)=Uniform[18C,26C](x) expresses the belief that the temperature at noon is distributed uniformly between 18 and 26 degrees Celsius. We call this a **probability density function.**
P(Weather , Cavity) denotes the probabilities of all combinations of the values of Weather and Cavity. This is a 4 × 2 table of probabilities called the **joint probability distribution** of Weather and Cavity.
For example, the product rules for all possible values of Weather and Cavity can be written as a single equation:

P(Weather , Cavity)=P(Weather | Cavity)P(Cavity) ,

<u>Probability axioms and their reasonableness</u>

The basic axioms of probability imply certain relationships among the degrees of belief that can be accorded to logically related propositions. We think a proposition a as the event A (set of sample points) where the proposition is true

- *Odd* is a propositional random variable of range {*true, false*}
- notation: $a \Leftrightarrow$ "$A = true$"
- Given Boolean random variables A and B:
- a: set of sample points where $A(\omega) = true$
- $\neg a$: set of sample points where $A(\omega) = false$
- $a \wedge b$: set of sample points where $A(\omega) = true$, $B(\omega) = true$
- with Boolean random variables, sample points are PL models Proposition: disjunction of the sample points in which it is true
  ex: $(a \vee b) \equiv (\neg a \wedge b) \vee (a \wedge \neg b) \vee (a \wedge b)$
  $P(a \vee b) = P(\neg a \wedge b) + P(a \wedge \neg b) + P(a \wedge b)$
- Some derived facts:
  $P(\neg a) = 1 - P(a)$

We can also derive the well known formula for the probability of a disjunction, sometimes called the inclusion–exclusion principle:

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

5	a	**Explain knowledge acquisition in detail.** **Answer:- (5M)** **Overview** Knowledge acquisition is the process of extracting domain-specific knowledge from experts to build expert systems. This involves converting expert insights into rules that machines can use for problem-solving. It is traditionally manual, expensive, and time-consuming, but automated or semi-automated tools (like MOLE and SALT) aim to streamline this process. **Key Activities Supported by Knowledge Acquisition Programs** 1. **Entering Knowledge:** Capturing domain knowledge systematically. 2. **Maintaining Consistency:** Ensuring logical consistency within the knowledge base. 3. **Ensuring Completeness:** Filling gaps to ensure the system can handle a wide range of problems. These programs typically work best when applied to well-defined problem paradigms like **diagnosis** or **design**. **MOLE: A Knowledge Acquisition System** ● **Purpose:** Developed for heuristic classification, especially in disease diagnosis. ● **Method:** Uses the **cover-and-differentiate** strategy. ○ First, **covering knowledge** helps identify all potential explanations for symptoms. ○ Then, **differentiating knowledge** is used to choose the best explanation. ○ The system refines its knowledge base iteratively through expert feedback. **MOLE-p (MOLE-Performance)** Used to solve real-world problems using the knowledge acquired. The acquisition process includes: 1. **Initial Knowledge Base Construction:** ○ MOLE collects symptoms, generates possible explanations, and builds causal chains (influence networks). 2. **Refinement:** ○ Identifies gaps in knowledge by solving example cases and prompting the expert to fill in missing rules. **SALT: Propose-and-Revise Strategy**

- **Focus:** Used in design tasks (e.g., elevator systems) where multiple parameters and constraints exist.
- **Method:** Builds a **dependency network** using:
    1. `contributes-to`: procedures generating values.
    2. `constrains`: restricts parameter values.
    3. `suggests-revision-of`: suggests changes when constraints are violated.
- SALT uses heuristics to:
    1. Ensure all parameters have contributing links.
    2. Prevent feedback loops.
    3. Add revision suggestions to constraint links.
- SALT turns the dependency network into production rules and can simulate problem-solving with expert oversight.

Meta-DENDRAL: Rule Induction via Learning

- First system to use **machine learning** for expert system rule generation.
- Used **version space learning** to analyze mass spectrometry data and determine molecular structures.
- Showcased that rules can be induced from data instead of solely relying on experts.

Challenges and Observations

- Experts are often **inarticulate** about their knowledge.
- **Statistical methods** may find patterns but lack **explainability**, making them unsuitable for expert systems requiring transparent decision-making.
- Machine learning is helpful but inadequate for tasks needing **causal reasoning**.

b

Explain independence and quantifying uncertainty with an example.

INDEPENDENCE

Expand the full joint distribution in Figure 13.3 by adding a fourth variable, **Weather**. The full joint distribution then becomes P(Toothache , Catch, Cavity, Weather), which has $2 \times 2 \times 2 \times 4 = 32$ entries. It contains four "editions" of the table shown in Figure 13.3, one for each kind of weather. What relationship do these editions have to each other and to the original three-variable table? For example, how are P (toothache , catch , cavity , cloudy) and P (toothache , catch , cavity) related? We can use the product rule:

P (toothache , catch , cavity , cloudy) = P (cloudy | toothache , catch , cavity)P (toothache , catch , cavity)

the weather does not influence the dental variables. Therefore, the following assertion seems reasonable:

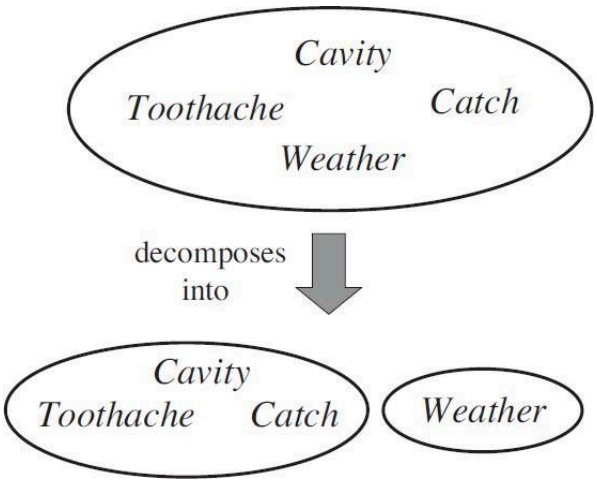
P (cloudy | toothache , catch , cavity)=P (cloudy) .
From this, we can deduce

P (toothache , catch , cavity , cloudy)=P (cloudy)P (toothache , catch , cavity) .

we can write the general equation

P(Toothache, Catch, Cavity, Weather)=P(Toothache , Catch, Cavity)P(Weather) .

The 32-element table for four variables can be constructed from one 8-element table and one 4-element table. This decomposition is illustrated schematically in Figure below.



The weather is independent of one's dental problems. Independence between propositions a and b can be written as

$$P(a \mid b) = P(a) \quad \text{or} \quad P(b \mid a) = P(b) \quad \text{or} \quad P(a \wedge b) = P(a)P(b)$$

Independence between variables X and Y can be written as follows

$$\mathbf{P}(X \mid Y) = \mathbf{P}(X) \quad \text{or} \quad \mathbf{P}(Y \mid X) = \mathbf{P}(Y) \quad \text{or} \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y)$$

Let's consider an example of uncertain reasoning: diagnosing a dental patient's toothache. A medical diagnosis

- Given the symptoms (toothache) infer the cause (cavity) How to encode this relation in logic?
    - diagnostic rules:
    - *Toothache* → *Cavity* (wrong)
    - *Toothache* → (*Cavity* ∨ *GumProblem* ∨ *Abscess* ∨ ...) (too many possible causes, some very unlikely) causal rules:
    - *Cavity* → *Toothache* (wrong)
    - (*Cavity* ∧ ...) → *Toothache* (many possible (con)causes)
- Problems in specifying the correct logical rules: Complexity: too many possible antecedents or consequents Theoretical ignorance: no complete theory for the domain Practical ignorance: no complete knowledge of the patient

Trying to use logic to cope with a domain like medical diagnosis thus fails for three main reasons:

- **Laziness:** It is too much work to list the complete set of antecedents or consequents

needed to ensure an exceptionless rule and too hard to use such rules.

- **Theoretical ignorance:** Medical science has no complete theory for the domain.
- **Practical ignorance:** Even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.

The connection between toothaches and cavities is just not a logical consequence in either direction. This is typical of the medical domain, as well as most other judgmental domains: law, business, design, automobile repair, gardening, dating, and so on. The agent's knowledge can at best provide only a **degree of belief** in the relevant sentences. Our main tool for dealing with degrees of belief is **probability theory.** Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance, thereby solving the qualification problem.

- Probability allows to summarize the uncertainty on effects of laziness: failure to enumerate exceptions, qualifications, etc. ignorance: lack of relevant facts, initial conditions, etc.
- Probability can be derived from, statistical data (ex: 80% of toothache patients so far had cavities) some knowledge (ex: 80% of toothache patients has cavities) their combination.
- Probability statements are made with respect to a state of knowledge (aka evidence), not with respect to the real world
- e.g., "The probability that the patient has a cavity, given that she has a toothache, is 0.8":
- *P*(*HasCavity* (*patient*) | hasToothAche(patient)) = 0.8
- Probabilities of propositions change with new evidence:

- "The probability that the patient has a cavity, given that she has a toothache and a history of gum disease, is 0.4":
- *P*(*HasCavity* (*patient*) | *hasToothAche*(*patient*) A *HistoryOfGum*(*patient*)) = 0.4

6

Write the representation of Bayes Theorem. In a class, 70% children fall sick due to Viral fever and 30% due to Bacterial fever. The probability of observing temperature for Viral is 0.78 and for Bacterial is 0.31. If a child develops high temperature, find the child's probability of having viral infection.

Answer:- (5+5M)

Bayes' Theorem is represented mathematically as:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Where:

- P(H | E)P(H|E)P(H | E) = Posterior probability (probability of **hypothesis** HHH given evidence EEE)

- P(E | H)P(E|H)P(E | H) = Likelihood (probability of observing **evidence** EEE given HHH is true)

- P(H)P(H)P(H) = Prior probability of hypothesis HHH
- Let VVV = Viral fever
- Let BBB = Bacterial fever
- Let TTT = High temperature

We are given:

- P(V)=0.70P(V) = 0.70P(V)=0.70
- P(B)=0.30P(B) = 0.30P(B)=0.30
- P(T∣V)=0.78P(T|V) = 0.78P(T∣V)=0.78
- P(T∣B)=0.31P(T|B) = 0.31P(T∣B)=0.31

We want to find:

P(V∣T)=?P(V|T) = ?P(V∣T)=?

First, compute the total probability of high temperature $P(T)$:

$$P(T) = P(T|V) \cdot P(V) + P(T|B) \cdot P(B)$$

$$P(T) = (0.78 \cdot 0.70) + (0.31 \cdot 0.30) = 0.546 + 0.093 = 0.639$$

Now, apply Bayes' Theorem:

$$P(V|T) = \frac{P(T|V) \cdot P(V)}{P(T)} = \frac{0.78 \cdot 0.70}{0.639} = \frac{0.546}{0.639} \approx 0.854$$

P(Viral∣Temperature)≈0.854

CI **CCI** **HOD**

----------All the Best----------

CO-PO and CO-PSO Mapping

Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Apply knowledge of agent architecture, searching and reasoning techniques for different applications.	**L1**	**1**	3	3	3	2	2	-	-	-	-	-	-	2	3	2	2	2
CO2	Compare various Searching and Inferencing Techniques.	**L2,L3**	**2,3,4**	3	3	3	2	2	-	-	-	-	-	-	2	3	2	2	2
CO3	Develop knowledge base sentences using propositional logic and first order logic	**L2**	**3,4**	3	3	3	3	2	-	-	-	-	-	-	2	3	2	2	2
CO4	Describe the concepts of quantifying uncertainty	**L3**	**4**	3	3	3	3	3	-	-	-	-	-	-	2	3	2	2	3
CO5	Use the concepts of Expert Systems to build applications.	**L2, L3**	**4,5**	3	3	3	2	3	-	-	-	-	-	-	2	3	2	2	3