Question 1:

Constraint Satisfaction Problem (CSP) A Constraint Satisfaction Problem (CSP) is a type of problem defined by: Variables: A set of variables $X = \{ X 1 , X 2 , . . . , X n \}$ X={X 1,X 2,...,X n} Domains: Each variable $X i$ X ihas a domain $D i$ D iof possible values it can take. Constraints: A set of constraints C C that specify allowable combinations of values among variables. Goal: Find an assignment to all variables such that all constraints are satisfied. Example In a scheduling problem, variables could be time slots, and constraints could include no two classes at the same time for the same student. Types of Consistencies in CSP Node Consistency: A variable is node-consistent if all values in its domain satisfy its unary constraints. Arc Consistency: A variable X X is arc-consistent with variable Y Y if for every value of X X, there is some value in Y Y's domain that satisfies the binary constraint between X X and Y Y. Path Consistency: A pair of variables X X and Y Y is path-consistent with a third variable Z Z if for every assignment to X X and Y Y, there is an assignment to Z Z that satisfies the constraints on the paths $X \to Z$ X→Z and $Z \to Y$ Z→Y. k-Consistency: A CSP is k-consistent if for any set of $k - 1$ k−1 variables and a consistent assignment to them, a value can be assigned to any $k t h$ k th variable such that all k k-ary constraints are satisfied. Cryptarithmetic Problem: markdown Copy SEND + MORE ------- MONEY Step-by-Step Solution: Each letter represents a unique digit from 0 to 9. First, some constraints: There are 8 unique letters: S, E, N, D, M, O, R, Y M must be 1 (since adding two 4-digit numbers gives a 5-digit number starting with M) No number can start with zero (S ≠ 0, M ≠ 0) Using backtracking or CSP solver, one possible valid assignment is: markdown Copy 9567 + 1085 ------- 10652 Mapping: S = 9 E = 5 N = 6 D = 7 M = 1 O = 0 R = 8 Y = 2 ✅ All constraints satisfied and the addition is correct. Let me know if you want the CSP or backtracking algorithm steps used to solve this.

Answer 1:

Constraint Satisfaction Problem (CSP)

A **Constraint Satisfaction Problem (CSP)** is a type of problem defined by:

- **Variables**: A set of variables $X = \{X_1, X_2, ..., X_n\}$X = {X1, X2, ..., Xn}
- **Domains**: Each variable X_iXi has a domain D_iDi of possible values it can take.
- **Constraints**: A set of constraints CC that specify allowable combinations of values among variables.

Goal: Find an assignment to all variables such that all constraints are satisfied.

Example

In a scheduling problem, variables could be time slots, and constraints could include no two classes at the same time for the same student.

Types of Consistencies in CSP

1. **Node Consistency**:

   - A variable is node-consistent if all values in its domain satisfy its unary constraints.

2. **Arc Consistency**:

   - A variable X is arc-consistent with variable Y if for every value of X, there is some value in Y's domain that satisfies the binary constraint between X and Y.

3. **Path Consistency**:

   - A pair of variables X and Y is path-consistent with a third variable Z if for every assignment to X and Y, there is an assignment to Z that satisfies the constraints on the paths $X \rightarrow Z$ and $Z \rightarrow Y$.

4. **k-Consistency**:

   - A CSP is k-consistent if for any set of $k-1$ variables and a consistent assignment to them, a value can be assigned to any k^{th} variable such that all k-ary constraints are satisfied.

Cryptarithmetic Problem:

```markdown

  SEND
+ MORE
-------
 MONEY
```

Step-by-Step Solution:

Each letter represents a unique digit from 0 to 9. First, some constraints:

- There are 8 unique letters: S, E, N, D, M, O, R, Y
- M must be 1 (since adding two 4-digit numbers gives a 5-digit number starting with M)
- No number can start with zero (S ≠ 0, M ≠ 0)

Using backtracking or CSP solver, one possible valid assignment is:

```
markdown

   9567
+ 1085
-------
 10652
```

Mapping:

- S = 9
- E = 5
- N = 6
- D = 7
- M = 1
- O = 0
- R = 8
- Y = 2

✅ All constraints satisfied and the addition is correct.

Let me know if you want the CSP or backtracking algorithm steps used to solve this.

Question 2:

Great question! Let's walk through how to solve the SEND + MORE = MONEY cryptarithmetic puzzle step-by-step — just like a Constraint Satisfaction Problem (CSP), using logical reasoning and constraint propagation (not guessing random numbers). 🧠 Step-by-Step Breakdown We write the equation: mathematica Copy S E N D + M O R E ---------- M O N E Y There are 8 unique letters: S, E, N, D, M, O, R, Y. Each letter must represent a different digit (0–9), and: S and M cannot be 0 (no leading zeros). The sum of SEND and MORE must exactly equal MONEY. Step 1: First digit analysis (leftmost column) The sum of two 4-digit numbers (SEND + MORE) gives a 5-digit number (MONEY). Hence, M = 1, because that's the only possible digit for the first digit of a 5-digit number produced from adding two 4-digit numbers. ✅ So: M = 1 Step 2: Column-by-column addition (from right to left) Let's rewrite the addition and align columns: mathematica Copy S E N D + 1 O R E ---------- 1 O N E Y We label column-wise from right to left: D + E = Y or (D + E + carry) % 10 = Y N + R + carry = E (from previous) E + O + carry = N S + 1 + carry = O First digit is M = 1 → already known. We'll now try to propagate constraints logically and try possible digits using backtracking. Step 3: Use known constraints From trial and constraint propagation (and computer-aided CSP backtracking, if implemented), we find that: ini Copy M = 1 O = 0 N = 6 E = 5 Y = 2 R = 8 D = 7 S = 9 Let's plug it in: markdown Copy SEND = 9567 MORE = 1085 -------------- MONEY = 10652 Verify the result: 9567 + 1085 = 10652 ✅ 🔁 Why this works This solution was found by: Assigning digits to letters such that no repetition occurs. Making sure each column of the addition respects the rules of base-10 math (carry over digits). Applying backtracking: if a constraint is violated, go back and try a different value. If you'd like, I can also: Show Python code to solve this using backtracking or constraint programming. Animate or diagram the step-by-step logic tree. Let me know!

Answer 2:

Great question! Let's walk through how to solve the **SEND + MORE = MONEY** cryptarithmetic puzzle step-by-step — just like a Constraint Satisfaction Problem (CSP), using logical reasoning and constraint propagation (not guessing random numbers).

🧠 Step-by-Step Breakdown

We write the equation:

```mathematica
   S E N D
+  M O R E
```

```
----------
 M O N E Y
```

There are **8 unique letters**: S, E, N, D, M, O, R, Y.
Each letter must represent a **different digit (0–9)**, and:

- **S and M cannot be 0** (no leading zeros).
- The sum of SEND and MORE must exactly equal MONEY.

Step 1: First digit analysis (leftmost column)

The sum of two 4-digit numbers (SEND + MORE) gives a 5-digit number (MONEY).
Hence, **M = 1**, because that's the only possible digit for the first digit of a 5-digit number produced from adding two 4-digit numbers.

✅ So: `M = 1`

Step 2: Column-by-column addition (from right to left)

Let’s rewrite the addition and align columns:

```
mathematica

   S E N D
+  1 O R E
----------
 1 O N E Y
```

We label column-wise from right to left:

1. **D + E = Y** or **(D + E + carry) % 10 = Y**
2. **N + R + carry = E** (from previous)
3. **E + O + carry = N**

4. **S + 1 + carry = O**

5. First digit is M = 1 → already known.

We'll now try to **propagate constraints** logically and try possible digits using **backtracking**.

Step 3: Use known constraints

From trial and constraint propagation (and computer-aided CSP backtracking, if implemented), we find that:

```ini
 M = 1
O = 0
N = 6
E = 5
Y = 2
R = 8
D = 7
S = 9
```

Let’s plug it in:

```markdown
   SEND  = 9567
  MORE  = 1085
--------------
 MONEY  = 10652
```

Verify the result:
9567 + 1085 = **10652** ✅

🔁 Why this works

This solution was found by:

- Assigning digits to letters such that no repetition occurs.

- Making sure each column of the addition respects the rules of base-10 math (carry over digits).
- Applying backtracking: if a constraint is violated, go back and try a different value.

If you'd like, I can also:

- Show Python code to solve this using backtracking or constraint programming.
- Animate or diagram the step-by-step logic tree.

Let me know!

Question 3:

Perfect! Let's go through the SEND + MORE = MONEY cryptarithmetic problem step-by-step — manually and logically — just like how a human or a backtracking CSP solver would reason. Step 1: Write the Problem Clearly mathematica Copy S E N D + M O R E ----------- M O N E Y Each letter stands for a unique digit from 0–9. No two letters can represent the same digit. Also, S ≠ 0 and M ≠ 0 (because numbers cannot start with 0). So, the unique letters are: S, E, N, D, M, O, R, Y → 8 letters → must map to 8 different digits. Step 2: Start With Most Constrained Information (Leftmost Column) We are adding two 4-digit numbers and getting a 5-digit number: nginx Copy SEND + MORE = MONEY So the first digit M must be 1 (since it's the only digit that can create a 5-digit number from two 4-digit numbers): ✅ M = 1 Step 3: Column-by-Column Addition (from right to left) We write it column-wise with possible carries. Let's align it: mathematica Copy S E N D + 1 O R E ---------------- 1 O N E Y We'll analyze from rightmost column to left. Column 1: D + E = Y (with possible carry) Let's say there's a carry C 1 C 1from this column to the next. mathematica Copy D + E = Y (mod 10) C_1 = ⌊(D + E)/10⌋ We don't know D, E, Y yet — so we defer until we get more constraints. Column 2: N + R + C_1 = E (with carry C_2) mathematica Copy N + R + C1 = E (mod 10) C2 = ⌊(N + R + C1)/10⌋ Again, too many unknowns — wait. Column 3: E + O + C2 = N (mod 10) Same situation — defer. Column 4: S + 1 + C3 = O (mod 10) Important! This one has more knowns: We know M = 1 So we know the leftmost digit of MORE is 1 So in this column: csharp Copy S + 1 + C3 = O (mod 10) C4 = carry out = ⌊(S + 1 + C3)/10⌋ We also know that: Leftmost digit of MONEY is M = 1 So this column produces a carry of 1 → so C4 = 1 So: mathematica Copy S + 1 + C3 = O + 10 → because we carry 1 We still don't know C3, O, S — but we can try possible values. Step 4: Try with Backtracking Let's try: M = 1 Try O = 0 (most likely candidate) Now try values for the rest in a consistent way, testing each column Try: ini Copy M = 1 O = 0 Now, try Column 4: java Copy S + 1 + C3 = 0 + 10 → S + 1 + C3 = 10 → S + C3 = 9 → Try C3 = 0 → S = 9 ✅ (possible) Now: S = 9 M = 1 O = 0 Try Column 3: mathematica Copy E + O + C2 = N (mod 10) E + 0 + C2 = N Try E = 5, N = 6 → 5 + 0 = 5 → C2 = 0 → N = 5 ❌ conflict Try E = 5, N = 6 → 5 + 0 = 5 → not 6 ❌ Try E = 5, C2 = 1 → 5 + 0 + 1 = 6 → N = 6 ✅ So try: - E = 5 - N = 6 So far: M = 1 O = 0 S = 9 E = 5 N = 6 Try Column 2: vbnet Copy N + R

+ C1 = E 6 + R + C1 = 5 (mod 10) Try R = 8, C1 = 1 → 6 + 8 + 1 = 15 → 15 mod 10 = 5 ✅ So: - R = 8 - C1 = 1 Try Column 1: mathematica Copy D + E = Y (mod 10) E = 5 C1 = 1 D + 5 = Y + 10 → D = Y + 5 Try Y = 2 → D = 7 ✅ (fits and unused) So: - D = 7 - Y = 2 Now we have: ini Copy M = 1 O = 0 N = 6 E = 5 Y = 2 R = 8 D = 7 S = 9 All digits are unique and between 0–9 ✅ Step 5: Verify the Full Addition SEND = 9567 MORE = 1085 MONEY = 10652 Check: 9567 + 1085 = 10652 ✅ ✅ Final Answer (Derived Logically): M = 1 O = 0 N = 6 E = 5 Y = 2 R = 8 D = 7 S = 9 Would you like a Python implementation of this CSP/backtracking solution?

Answer 3:

Perfect! Let's go through the **SEND + MORE = MONEY** cryptarithmetic problem step-by-step — manually and logically — just like how a human or a backtracking CSP solver would reason.

Step 1: Write the Problem Clearly

```mathematica
   S E N D
+  M O R E
-----------
 M O N E Y
```

Each letter stands for a unique digit from 0–9.
No two letters can represent the same digit.
Also, **S ≠ 0** and **M ≠ 0** (because numbers cannot start with 0).

So, the unique letters are:
S, E, N, D, M, O, R, Y → 8 letters → must map to 8 different digits.

Step 2: Start With Most Constrained Information (Leftmost Column)

We are adding two 4-digit numbers and getting a 5-digit number:

```nginx
 SEND + MORE = MONEY
```

So the first digit **M** must be 1 (since it's the only digit that can create a 5-digit number from two 4-digit numbers):

- ✅ **M = 1**

Step 3: Column-by-Column Addition (from right to left)

We write it column-wise with possible carries.

Let's align it:

```
mathematica

     S  E  N  D
+   1  O  R  E
----------------
  1  O  N  E  Y
```

We'll analyze from rightmost column to left.

Column 1: D + E = Y (with possible carry)

Let's say there's a carry C_1C1 from this column to the next.

```
mathematica

 D + E = Y  (mod 10)
C_1 = ⌊(D + E)/10⌋
```

We don't know D, E, Y yet — so we defer until we get more constraints.

Column 2: N + R + C_1 = E (with carry C_2)

```
mathematica

 N + R + C1 = E (mod 10)
C2 = ⌊(N + R + C1)/10⌋
```

Again, too many unknowns — wait.

Column 3: E + O + C2 = N (mod 10)

Same situation — defer.

Column 4: S + 1 + C3 = O (mod 10)

Important! This one has more knowns:

- We know M = 1
- So we know the leftmost digit of MORE is 1
- So in this column:

  ```
  csharp

   S + 1 + C3 = O  (mod 10)
  C4 = carry out = ⌊(S + 1 + C3)/10⌋
  ```

We also know that:

- Leftmost digit of MONEY is M = 1
- So this column produces a carry of 1 → so C4 = 1

So:

```
mathematica

 S + 1 + C3 = O + 10  → because we carry 1
```

We still don’t know C3, O, S — but we can try **possible values**.

Step 4: Try with Backtracking

Let’s try:

- M = 1

- Try O = 0 (most likely candidate)
- Now try values for the rest in a consistent way, testing each column

Try:

```ini
 M = 1
O = 0
```

Now, try **Column 4**:

```java
 S + 1 + C3 = 0 + 10
→ S + 1 + C3 = 10
→ S + C3 = 9
→ Try C3 = 0 → S = 9 ✅ (possible)
```

Now:

- S = 9
- M = 1
- O = 0

Try **Column 3**:

```mathematica
 E + O + C2 = N  (mod 10)
E + 0 + C2 = N

Try E = 5, N = 6 → 5 + 0 = 5 → C2 = 0 → N = 5 ❌ conflict
Try E = 5, N = 6 → 5 + 0 = 5 → not 6 ❌

Try E = 5, C2 = 1 → 5 + 0 + 1 = 6 → N = 6 ✅

So try:
```

```
- E = 5
- N = 6
```

So far:

- M = 1
- O = 0
- S = 9
- E = 5
- N = 6

Try **Column 2**:

```vbnet
 N + R + C1 = E
6 + R + C1 = 5 (mod 10)

Try R = 8, C1 = 1 → 6 + 8 + 1 = 15 → 15 mod 10 = 5 ✅

So:
- R = 8
- C1 = 1
```

Try **Column 1**:

```mathematica
 D + E = Y (mod 10)
E = 5
C1 = 1
D + 5 = Y + 10

→ D = Y + 5

Try Y = 2 → D = 7 ✅ (fits and unused)
So:
```

```
- D = 7
- Y = 2
```

Now we have:

```ini
 M = 1
O = 0
N = 6
E = 5
Y = 2
R = 8
D = 7
S = 9
```

All digits are unique and between 0–9 ✅

Step 5: Verify the Full Addition

SEND = 9567
MORE = 1085
MONEY = 10652

Check: 9567 + 1085 = 10652 ✅

✅ Final Answer (Derived Logically):

- M = 1
- O = 0
- N = 6
- E = 5
- Y = 2
- R = 8

- D = 7
- S = 9

Would you like a Python implementation of this CSP/backtracking solution?

Question 4:

Here is the full 10-mark theoretical answer extracted directly from your provided textbook (Ai_unit_3.pdf) on the Minimax Search Algorithm for two players and how alpha-beta pruning improves its performance. ✦ Minimax Algorithm for Two Players Mini-Max Algorithm The minimax algorithm computes the minimax decision from the current state. It is used as a searching technique in game problems. The minimax algorithm performs a complete depth-first exploration of the game-tree. The Algorithm: The start node is MAX (player 1) node with current board configuration. Expand nodes down (play) to some depth of look-ahead in the game. Apply evaluation function at each of the leaf nodes. “Back up" values for each non-leaf node until computed for the root node. At MIN (player 2) nodes, the backed-up value is the minimum of the values associated with its children. At MAX nodes, the backed-up value is the maximum of the values associated with its children. Note: The process of “backing up” values gives the optimal strategy, that is, both players assume that your opponent is using the same static evaluation function as you are . ✦ Use of Alpha-Beta Cut-Offs to Improve Performance Alpha-beta cut-offs is a modified version of minimax algorithm, wherein two threshold values are maintained for future expansion. One representing a lower bound on the value that a maximizing node ultimately can be assigned (we call this alpha) Another representing an upper bound on the value that a minimizing node may be assigned (this we call beta) Working: At each node, alpha and beta values may be updated as we iterate over the node's children. If it's a maximizer node, then alpha is increased if a child value is greater than the current alpha. At a minimizer node, beta may be decreased. Pruning happens when alpha ≥ beta, i.e., when the alpha and beta lines hit each other in the node value. This means the remaining child nodes need not be explored, as they won't affect the outcome. This significantly reduces the number of nodes evaluated, improving performance. Example: The maximizer assigns a value 9 at B (minimum of D and E), which is passed back to A. So the maximizer is assured of value ≥ 9. Now, if C leads to a value of 5 or less, A will never choose C — so we prune that branch. This saves a lot of time in searching. Hence, alpha-beta pruning is a very effective optimization for the minimax algorithm . ✦ Alpha-Beta Pruning Algorithm (from the book) Algorithm of Alpha-Beta Pruning vbnet Copy alpha is the best score for max along the path to state. beta is the best score for min along the path to state. If the level is the top level, let alpha = — infinity, beta = + infinity. If depth has reached the search limit, apply static evaluation function to state and return result. If player is max: Until all of state's children are examined with ALPHA-BETA or until alpha ≥ beta: Call ALPHA-BETA(child, min, depth + 1, alpha, beta); Compare the result with alpha; if result > alpha, reset alpha to the new value. Report alpha If player is min: Until all of state's children are examined with ALPHA-BETA or until alpha ≥ beta: Call ALPHA-BETA(child, max, depth + 1, alpha, beta); Compare the result with beta; if result < beta, reset beta to the new value. Report beta This algorithm efficiently eliminates large parts of the search tree which do not affect the final decision, making the game-playing agent faster and smarter . Let me know if you'd like a flowchart or implementation of this algorithm.

Answer 4:

Here is the **full 10-mark theoretical answer** extracted directly from your provided textbook (`Ai_unit_3.pdf`) on the **Minimax Search Algorithm for two players** and how **alpha-beta pruning** improves its performance.

✦ Minimax Algorithm for Two Players

> **Mini-Max Algorithm**
> The minimax algorithm computes the minimax decision from the current state. It is used as a searching technique in game problems. The minimax algorithm performs a complete depth-first exploration of the game-tree.
>
> **The Algorithm:**
>
> 1. The start node is MAX (player 1) node with current board configuration.
> 2. Expand nodes down (play) to some depth of look-ahead in the game.
> 3. Apply evaluation function at each of the leaf nodes.
> 4. “Back up" values for each non-leaf node until computed for the root node.
> 5. At MIN (player 2) nodes, the backed-up value is the **minimum** of the values associated with its children.
> 6. At MAX nodes, the backed-up value is the **maximum** of the values associated with its children.
>
> **Note:** The process of “backing up” values gives the optimal strategy, that is, both players assume that your opponent is using the same static evaluation function as you are.

✦ Use of Alpha-Beta Cut-Offs to Improve Performance

Alpha-beta cut-offs is a modified version of minimax algorithm, wherein **two threshold values** are maintained for future expansion.

- One representing a **lower bound** on the value that a **maximizing node** ultimately can be assigned (we call this **alpha**)
- Another representing an **upper bound** on the value that a **minimizing node** may be assigned (this we call **beta**)

Working:
At each node, alpha and beta values may be updated as we iterate over the node's children.

- If it's a maximizer node, then **alpha is increased** if a child value is greater than the current alpha.
- At a minimizer node, **beta may be decreased**.

Pruning happens when **alpha ≥ beta**, i.e., when the alpha and beta lines hit each other in the node value.

This means the remaining child nodes need not be explored, as they won't affect the outcome. This significantly **reduces the number of nodes evaluated**, improving performance.

> Example:
> The maximizer assigns a value 9 at B (minimum of D and E), which is passed back to A. So the maximizer is assured of value ≥ 9.
> Now, if C leads to a value of 5 or less, A will never choose C — so we **prune** that branch.
> This saves a lot of time in searching.

Hence, **alpha-beta pruning** is a very effective optimization for the minimax algorithm.

✦ Alpha-Beta Pruning Algorithm (from the book)

Algorithm of Alpha-Beta Pruning

```vbnet

 alpha is the best score for max along the path to state.
beta is the best score for min along the path to state.

If the level is the top level, let alpha = – infinity, beta = + infinity.
If depth has reached the search limit, apply static evaluation function to state and return result.

If player is max:
    Until all of state's children are examined with ALPHA-BETA or until alpha ≥ beta:
        Call ALPHA-BETA(child, min, depth + 1, alpha, beta);
        Compare the result with alpha; if result > alpha, reset alpha to the new value.
    Report alpha

If player is min:
    Until all of state's children are examined with ALPHA-BETA or until alpha ≥ beta:
        Call ALPHA-BETA(child, max, depth + 1, alpha, beta);
        Compare the result with beta; if result < beta, reset beta to the new value.
    Report beta
```

This algorithm efficiently eliminates large parts of the search tree which do not affect the final decision, making the game-playing agent faster and smarter.

Let me know if you'd like a flowchart or implementation of this algorithm.

Question 5:

Here is the full 10-mark theoretical answer on Game Theory, including the difference between stochastic and partial games with examples, exactly as extracted from your provided textbook Ai_unit_3.pdf: ✦ Definition of Game Theory Game theory denotes games of strategy. John Von Neumann is acknowledged as father of game theory. Neumann defined game theory in 1928 and established the mathematical framework for all subsequent theoretical developments. Game theory allows decision-makers (players) to cope with other decision-makers (players) who have different purposes in mind. In other words, players determine their own strategies in terms of the strategies and goal of their opponent. Games are integral attribute of human beings. Games engage the intellectual faculties of humans. If computers are to mimic people they should be able to play games. Game playing has close relation to intelligence and it has well-defined states and rules . Game theory does not prescribe a way or say how to play a game. Game theory is a set of ideas and techniques for analyzing conflict situations between two or more parties . ✦ Stochastic Games (Definition + Example) A stochastic game is a collection of normal-form games that the agents play repeatedly. The particular game played at any time depends probabilistically on the previous game played and the actions of the agents in that game. Like a probabilistic finite state machine in which the states are the games and the transition labels are joint action-payoff pairs. A stochastic game also called a Markov game is defined by: a finite set Q of states (games), a set of strategies Si(x) for each player for each state x ∈ X, a set N = {1, ..., n} of agents, for each agent i, a finite set Ai of possible actions, a transition probability function a set of rewards dependent on the state and the actions of the other players . Example: One example of a two-player zero-sum stochastic game is Backgammon. Two agents take turns. Before his/her move, an agent must roll the dice. The set of available moves depends on the results of the dice roll. Each move has a stochastic outcome, due to the dice roll . ✦ Partially Observable Games (Definition + Example) Partial observability means that an agent does not know the state of the world or that the agents act simultaneously. A partially observable system is one in which the entire state of the system is not fully visible to an external sensor. The observer may utilize a memory system to enhance decision making based on past observations . Example: A card game where some cards are discarded face down. The observer is only able to view their own cards and possibly those of the dealer. They cannot see the face-down cards or those that will be dealt in the future. This restricts the information available and makes the game partially observable . ✦ Difference Between Stochastic and Partial Games Feature Stochastic Games Partial Games Definition Game where transitions depend on probabilities and player actions Game where not all information about the state is visible to all players Core Complexity Involves randomness in transitions between states Involves hidden or limited information per agent Transition Mechanism Probabilistic transitions between game states Uncertainty due to partial visibility of the game environment Example Backgammon — moves depend on dice outcomes Poker/Card Games — where players cannot see others' hands Evaluation Method Uses algorithms like Expectiminimax, stochastic modeling Uses belief states, memory systems, and depth-limited reasoning Application Areas Economics, resource allocation, operations research Strategic games, security, multi-agent systems Let me know if you want these notes formatted for exam writing or want help with expected diagrams for this topic.

Answer 5:

Here is the full **10-mark theoretical answer** on **Game Theory**, including the **difference between stochastic and partial games with examples**, exactly as extracted from your provided textbook *Ai_unit_3.pdf*:

✦ Definition of Game Theory

> Game theory denotes games of strategy.
> John Von Neumann is acknowledged as father of game theory. Neumann defined game theory in 1928 and established the mathematical framework for all subsequent theoretical developments.
> Game theory allows decision-makers (players) to cope with other decision-makers (players) who have different purposes in mind. In other words, players determine their own strategies in terms of the strategies and goal of their opponent.
> Games are integral attribute of human beings. Games engage the intellectual faculties of humans.

If computers are to mimic people they should be able to play games. Game playing has close relation to intelligence and it has well-defined states and rules.

Game theory does not prescribe a way or say how to play a game. Game theory is a set of ideas and techniques for analyzing conflict situations between two or more parties.

✦ Stochastic Games (Definition + Example)

A stochastic game is a collection of normal-form games that the agents play repeatedly. The particular game played at any time depends probabilistically on the previous game played and the actions of the agents in that game.
Like a probabilistic finite state machine in which the states are the games and the transition labels are joint action-payoff pairs.
A stochastic game also called a **Markov game** is defined by:

- a finite set Q of states (games),
- a set of strategies $S_i(x)$ for each player for each state $x \in X$,
- a set $N = \{1, ..., n\}$ of agents,
- for each agent i, a finite set A_i of possible actions,
- a transition probability function
- a set of rewards dependent on the state and the actions of the other players.

Example:
One example of a two-player zero-sum stochastic game is **Backgammon**.

- Two agents take turns. Before his/her move, an agent must roll the dice.
- The set of available moves depends on the results of the dice roll.
- Each move has a stochastic outcome, due to the dice roll.

✦ Partially Observable Games (Definition + Example)

Partial observability means that an agent does not know the state of the world or that the agents act simultaneously.
A partially observable system is one in which the entire state of the system is not fully visible to an external sensor.
The observer may utilize a memory system to enhance decision making based on past observations.

Example:
A **card game** where some cards are discarded face down.

- The observer is only able to view their own cards and possibly those of the dealer.
- They cannot see the face-down cards or those that will be dealt in the future.
- This restricts the information available and makes the game **partially observable**.

✦ Difference Between Stochastic and Partial Games

Feature	Stochastic Games	Partial Games
Definition	Game where transitions depend on probabilities and player actions	Game where not all information about the state is visible to all players
Core Complexity	Involves randomness in transitions between states	Involves hidden or limited information per agent
Transition Mechanism	Probabilistic transitions between game states	Uncertainty due to partial visibility of the game environment
Example	**Backgammon** — moves depend on dice outcomes	**Poker/Card Games** — where players cannot see others' hands
Evaluation Method	Uses algorithms like **Expectiminimax**, stochastic modeling	Uses belief states, memory systems, and depth-limited reasoning
Application Areas	Economics, resource allocation, operations research	Strategic games, security, multi-agent systems

---

Let me know if you want these notes formatted for exam writing or want help with expected diagrams for this topic.

Question 6:

Let's go step by step to solve the full 8-mark question: ✅ 1. Define Constraint Satisfaction Problem (CSP) A Constraint Satisfaction Problem (CSP) is a mathematical problem defined as a triple (X , D , C) (X,D,C) where: X is a set of variables: { X 1 , X 2 , . . . , X n } {X 1,X 2,...,X n} D is a set of domains: each variable X_i X ihas a domain D_i D iof possible values. C is a set of constraints: conditions that must be satisfied between variables. Goal: Assign a value to every variable from its domain such that all the constraints are satisfied. Example: Cryptarithmetic puzzles, Sudoku, Map Coloring, Scheduling. ✅ 2. Types of Consistencies in CSP Type Description Node Consistency A variable is node-consistent if all values in its domain satisfy unary constraints. Arc Consistency A variable X X is arc-consistent with Y Y if for every value in X X, there exists a consistent value in Y Y. Path Consistency A path between variables X X and Z Z via Y Y is consistent if for every pair of values of X X and Z Z, there's a Y Y value satisfying both constraints. k-Consistency A CSP is k-consistent if any consistent assignment to $k-1$ k−1 variables can be extended to a kth k th variable. ✅ 3. Solve the Cryptarithmetic Problem Problem: css Copy B A S E + B A L L --------- G A M E S Each letter stands for a unique digit (0–9), and no number starts with 0. Step 1: Count the unique letters Letters involved: B, A, S, E, L, G, M Total unique letters = 7 → must be assigned digits 0–9 with no repetition. Step 2: Make Observations G must be 1 because the sum of two 4-digit numbers results in a 5-digit number (GAMES). ✅ So: G = 1 Let's try a valid digit assignment using constraints and check if the sum works. Step 3: Try known valid mapping (based on backtracking with constraints): ini Copy B = 7 A = 4 S = 8 E = 3 L = 5 G = 1 M = 9 Now verify the numbers: java Copy B A S E = 7483 + B A L L = 7455 ------------- G A M E S = 14938 ✅ 7483 + 7455 = 14938 Correct! ✅ Final Answer: ➤ Letter → Digit mapping: B = 7 A = 4 S = 8 E = 3 L = 5 G = 1 M = 9 ➤ Numbers: markdown Copy BASE = 7483 + BALL = 7455 ------------- GAMES = 14938 ✅ Would you like to see how to program this step-by-step with backtracking or CSP solver logic?

Answer 6:

Let's go step by step to solve the full 8-mark question:

✅ 1. Define Constraint Satisfaction Problem (CSP)

A **Constraint Satisfaction Problem (CSP)** is a mathematical problem defined as a triple (X, D, C) (X, D, C) where:

- **X** is a set of variables: $\{X_1, X_2, ..., X_n\}${X1, X2, ..., Xn}
- **D** is a set of domains: each variable X_iXi has a domain D_iDi of possible values.
- **C** is a set of constraints: conditions that must be satisfied between variables.

Goal: Assign a value to every variable from its domain such that all the constraints are satisfied.

Example: Cryptarithmetic puzzles, Sudoku, Map Coloring, Scheduling.

✅ 2. Types of Consistencies in CSP

Type	Description
Node Consistency	A variable is node-consistent if all values in its domain satisfy unary constraints.
Arc Consistency	A variable X is arc-consistent with Y if for every value in X, there exists a consistent value in Y.
Path Consistency	A path between variables X and Z via Y is consistent if for every pair of values of X and Z, there's a Y value satisfying both constraints.
k-Consistency	A CSP is k-consistent if any consistent assignment to $k-1$ variables can be extended to a kth variable.

✅ 3. Solve the Cryptarithmetic Problem

Problem:

```css
   B A S E
+ B A L L
---------
G A M E S
```

Each letter stands for a unique digit (0–9), and no number starts with 0.

Step 1: Count the unique letters

Letters involved: **B, A, S, E, L, G, M**

- Total unique letters = 7 → must be assigned digits 0–9 with no repetition.

Step 2: Make Observations

- G must be **1** because the sum of two 4-digit numbers results in a 5-digit number (GAMES).
  ✅ So: `G = 1`

Let's try a valid digit assignment using constraints and check if the sum works.

Step 3: Try known valid mapping (based on backtracking with constraints):

```ini
 B = 7
A = 4
S = 8
E = 3
L = 5
G = 1
M = 9
```

Now verify the numbers:

```java
   B A S E = 7483
+ B A L L = 7455
-------------
G A M E S = 14938
```

✅ 7483 + 7455 = **14938**

Correct!

✅ Final Answer:

➤ Letter → Digit mapping:

- B = 7
- A = 4
- S = 8
- E = 3
- L = 5
- G = 1

- M = 9

➤ Numbers:

```
markdown

  BASE = 7483
+ BALL = 7455
-------------
GAMES = 14938 ✅
```

Would you like to see how to program this step-by-step with backtracking or CSP solver logic?

Question 7:

Here is the full 10-mark answer on Minimax and Alpha-Beta Pruning algorithms for adversarial search, exactly extracted from your provided textbook Ai_unit_3.pdf. ✅ Mini-Max Algorithm (From Textbook) The minimax algorithm computes the minimax decision from the current state. It is used as a searching technique in game problems. The minimax algorithm performs a complete depth-first exploration of the game-tree. The Algorithm: The start node is MAX (player 1) node with current board configuration. Expand nodes down (play) to some depth of look-ahead in the game. Apply evaluation function at each of the leaf nodes. “Back up" values for each non-leaf node until computed for the root node. At MIN (player 2) nodes, the backed-up value is the minimum of the values associated with its children. At MAX nodes, the backed-up value is the maximum of the values associated with its children. Note: The process of “backing up” values gives the optimal strategy, that is, both players assume that your opponent is using the same static evaluation function as you are . ✅ Properties of Mini-Max Minimax provides a complete solution for finite tree. Minimax provides optimal strategy against an optimal opponent. The time complexity is $O(b^m)$ O(b m). The space complexity is $O(bm)$ O(bm) for depth-first exploration where the algorithm generates all successors at once, or $O(m)$ O(m) for generating one at a time . ✅ Alpha-Beta Pruning Algorithm (From Textbook) Alpha-beta cut-offs is a modified version of the minimax algorithm, wherein two threshold values are maintained for future expansion: One representing a lower bound on the value that a maximizing node ultimately can be assigned (alpha) Another representing an upper bound on the value that a minimizing node may be assigned (beta) . ✅ Algorithm of Alpha-Beta Pruning text Copy alpha is the best score for max along the path to state. beta is the best score for min along the path to state. If the level is the top level, let alpha = -∞, beta = +∞. If depth has reached the search limit, apply static evaluation function to state and return result. If player is MAX: Until all of state's children are examined or alpha ≥ beta: Call ALPHA-BETA(child, MIN, depth + 1, alpha, beta) If result > alpha, reset alpha = result Report alpha If player is MIN: Until all of state's children are examined or alpha ≥ beta: Call ALPHA-BETA(child, MAX, depth + 1, alpha, beta) If result < beta, reset beta = result Report beta ✅ Example of Alpha-Beta Pruning In a game tree, each node represents a board position where one of the players gets to choose a move. For example, at node C, if the minimizer can limit utility to 2, and maximizer can get 6 from another branch (node B), he will avoid node C. Hence, no need to evaluate other children of C. At each node, alpha and beta values are updated. When alpha ≥ beta, pruning occurs. This saves time by eliminating nodes that don't affect the final decision . ✅ Diagram Reference in Book Fig. 4.12.6 and Fig. 4.12.8 show visual examples of pruning with updated alpha-beta values. The tree is explored only as far as needed to make the decision — without compromising correctness . ✅ Summary of Performance Gain Without pruning, time complexity is $O(b^m)$ O(b m). With perfect ordering, alpha-beta pruning reduces it to $O(b^{m/2})$ O(b m/2). This means twice the depth can be searched in the same time. Would you like a diagram redrawn from the book to support this in your notes?

Answer 7:

Here is the **full 10-mark answer** on **Minimax and Alpha-Beta Pruning algorithms for adversarial search**, *exactly extracted* from your provided textbook `Ai_unit_3.pdf`.

✅ Mini-Max Algorithm (From Textbook)

The **minimax algorithm** computes the minimax decision from the current state. It is used as a searching technique in game problems. The minimax algorithm performs a complete depth-first exploration of the game-tree.

The Algorithm:

1. The start node is MAX (player 1) node with current board configuration.

2. Expand nodes down (play) to some depth of look-ahead in the game.

3. Apply evaluation function at each of the leaf nodes.

4. "Back up" values for each non-leaf node until computed for the root node.

5. At MIN (player 2) nodes, the backed-up value is the **minimum** of the values associated with its children.

6. At MAX nodes, the backed-up value is the **maximum** of the values associated with its children.

Note: The process of "backing up" values gives the optimal strategy, that is, both players assume that your opponent is using the same static evaluation function as you are.

Properties of Mini-Max

1. Minimax provides a complete solution for finite tree.

2. Minimax provides optimal strategy against an optimal opponent.

3. The time complexity is $O(b^m)$O(bm).

4. The space complexity is $O(bm)$O(bm) for depth-first exploration where the algorithm generates all successors at once, or $O(m)$O(m) for generating one at a time.

Alpha-Beta Pruning Algorithm (From Textbook)

Alpha-beta cut-offs is a modified version of the minimax algorithm, wherein two threshold values are maintained for future expansion:

- One representing a lower bound on the value that a **maximizing** node ultimately can be assigned (**alpha**)

- Another representing an upper bound on the value that a **minimizing** node may be assigned (**beta**).

Algorithm of Alpha-Beta Pruning

```
text

 alpha is the best score for max along the path to state.
beta is the best score for min along the path to state.

If the level is the top level, let alpha = -∞, beta = +∞.
If depth has reached the search limit, apply static evaluation function to state and return result.

If player is MAX:
    Until all of state's children are examined or alpha ≥ beta:
        Call ALPHA-BETA(child, MIN, depth + 1, alpha, beta)
        If result > alpha, reset alpha = result
    Report alpha
```

```
If player is MIN:
    Until all of state's children are examined or alpha ≥ beta:
        Call ALPHA-BETA(child, MAX, depth + 1, alpha, beta)
        If result < beta, reset beta = result
    Report beta
```

✅ Example of Alpha-Beta Pruning

In a game tree, each node represents a board position where one of the players gets to choose a move.
For example, at node **C**, if the minimizer can limit utility to 2, and maximizer can get 6 from another branch (node **B**), he will avoid node **C**. Hence, no need to evaluate other children of **C**.

- At each node, **alpha** and **beta** values are updated.
- When **alpha ≥ beta**, pruning occurs.
- This saves time by **eliminating nodes that don't affect the final decision**.

✅ Diagram Reference in Book

- **Fig. 4.12.6** and **Fig. 4.12.8** show visual examples of pruning with updated alpha-beta values.
- The tree is explored **only as far as needed** to make the decision — **without compromising correctness**.

✅ Summary of Performance Gain

- Without pruning, time complexity is $O(b^m)$O(bm).
- With **perfect ordering**, **alpha-beta pruning** reduces it to $O(b^{m/2})$O(bm/2).
- This means **twice the depth can be searched in the same time**.

Would you like a diagram redrawn from the book to support this in your notes?

Question 8:

Here is the full 10-mark theoretical answer for the question "Define and explain Constraint Satisfaction Problem (CSP)", exactly extracted and organized from your provided textbook Ai_unit_3.pdf: ✅ Definition of Constraint Satisfaction Problem A constraint satisfaction problem (CSP) is defined by a set of variables $X_1, X_2, ..., X_n$ X 1,X 2,...,X nand a set of constraints $C_1, C_2, ..., C_m$ C 1,C 2,...,C m: Each variable X_i X ihas a non-empty domain D_i D iof possible values. Each constraint C_j C jinvolves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is assigned and a solution to a CSP is a complete assignment that satisfies all the constraints . ✅ Concept and Importance Constraint satisfaction problem has various states and a goal test; a traditional problem is converted into a structured and simple representation. General-purpose routines can be used to access this special representation, which offers more benefits than problem-specific heuristics. This structured approach allows: i) Problem decomposition ii) Understanding the structure and difficulty of the problem iii) Use of standard patterns (variables with assigned values), making it easy to design successor functions and goal tests applicable to all CSPs . ✅ Properties of Constraints Constraints may specify partial information and need not uniquely determine variable values. Constraints are non-directional: a constraint on variables V_1 V 1and V_2 V 2can be used to infer constraints in either direction. Constraints are declarative: they specify what relationship must hold, not how to enforce it. Constraints are additive: the order of applying constraints doesn't matter—only the conjunction matters. Constraints are independent: though they often share variables, they can be considered modular in behavior . ✅ Types of Constraints Unary Constraint: Restricts the value of a single variable. Example: Vertex A cannot be colored blue. Binary Constraint: Relates two variables. Example: In a graph coloring problem, adjacent vertices cannot have the same color. Higher-Order Constraints: Involve three or more variables. Example: Cryptarithmetic puzzles where each letter represents a unique digit . ✅ Examples of CSP Map Coloring: Assign colors to regions so that no adjacent regions share the same color. N-Queens Problem: Place queens on a chessboard such that no two attack each other. Sudoku: Assign digits to cells with row, column, and box constraints. Cryptarithmetic Problems: Assign digits to letters ensuring the arithmetic expression is valid . ✅ Real-World Applications Scheduling (e.g., telescope experiments, factory jobs) Scene labeling in vision Resource allocation Hardware configuration Linear programming and continuous-domain CSPs . Let me know if you'd like help converting this into an exam-ready handwritten format or diagrammatic form.

Answer 8:

Here is the **full 10-mark theoretical answer** for the question *"Define and explain Constraint Satisfaction Problem (CSP)"*, **exactly extracted** and organized from your provided textbook *Ai_unit_3.pdf*:

✅ Definition of Constraint Satisfaction Problem

A **constraint satisfaction problem (CSP)** is defined by a set of variables $X_1, X_2, ..., X_n$X1 ,X2, ..., Xn and a set of constraints $C_1, C_2, ..., C_m$C1, C2, ..., Cm:

- Each variable X_iXi has a non-empty domain D_iDi of possible values.
- Each constraint C_jCj involves some subset of the variables and specifies the allowable combinations of values for that subset.
- A **state** of the problem is defined by an assignment of values to some or all of the variables.
- An assignment that does not violate any constraints is called a **consistent or legal assignment**.
- A **complete assignment** is one in which every variable is assigned and a solution to a CSP is a complete assignment that satisfies all the constraints.

✅ Concept and Importance

- Constraint satisfaction problem has various states and a goal test; a traditional problem is converted into a structured and simple representation.
- General-purpose routines can be used to access this special representation, which offers more benefits than problem-specific heuristics.
- This structured approach allows:
  i) Problem decomposition
  ii) Understanding the structure and difficulty of the problem
  iii) Use of standard patterns (variables with assigned values), making it easy to design successor functions and goal tests applicable to all CSPs.

✅ Properties of Constraints

- Constraints may specify partial information and need not uniquely determine variable values.
- Constraints are **non-directional**: a constraint on variables V_1V1 and V_2V2 can be used to infer constraints in either direction.
- Constraints are **declarative**: they specify what relationship must hold, not how to enforce it.
- Constraints are **additive**: the order of applying constraints doesn't matter—only the conjunction matters.
- Constraints are **independent**: though they often share variables, they can be considered modular in behavior.

✅ Types of Constraints

1. **Unary Constraint**:
    - Restricts the value of a single variable.
    - Example: Vertex A cannot be colored blue.
2. **Binary Constraint**:
    - Relates two variables.
    - Example: In a graph coloring problem, adjacent vertices cannot have the same color.
3. **Higher-Order Constraints**:
    - Involve three or more variables.
    - Example: Cryptarithmetic puzzles where each letter represents a unique digit.

✅ Examples of CSP

- **Map Coloring**: Assign colors to regions so that no adjacent regions share the same color.
- **N-Queens Problem**: Place queens on a chessboard such that no two attack each other.
- **Sudoku**: Assign digits to cells with row, column, and box constraints.
- **Cryptarithmetic Problems**: Assign digits to letters ensuring the arithmetic expression is valid.

✅ Real-World Applications

- Scheduling (e.g., telescope experiments, factory jobs)
- Scene labeling in vision
- Resource allocation
- Hardware configuration
- Linear programming and continuous-domain CSPs.

Let me know if you'd like help converting this into an exam-ready handwritten format or diagrammatic form.

Question 9:

Here is the full 10-mark answer to the question "Explain with example graph coloring problem", exactly extracted from your provided textbook Ai_unit_3.pdf: ✅ Graph Coloring Problem (as explained in the book) Graph coloring problem is a type of constraint satisfaction problem (CSP). A constraint graph consists of: Nodes: Variables (e.g., vertices of the graph that need to be colored) Edges/Arcs: Constraints (e.g., adjacent vertices must not have the same color) ✦ Formal Definition and Example (Map Coloring as Graph Coloring) 5.1.3.1 Map Colouring Problem The problem is to colour the regions of a given map such that no 2 adjacent regions have the same colour. The regions in the map are the variables, and the set of possible colours for the regions is the domain. The constraint is that: “No two adjacent regions should have the same colour.” Formal Representation: Variables: {N, NW, NE, M, MW, ME, S, SE} Domains: D = {Red, Green, Blue} Constraints: Adjacent regions must have different colors Example: N ≠ NW We sometimes represent map colouring as a graph coloring (constraint graph) problem. Typically, such a problem has many solutions. The topology of a constraint graph can sometimes be used to identify solutions easily . ✦ Graph Coloring Example (with Figure Reference) In Fig. 5.1.2, a map of regions is shown, such as: North (N), Middle (M), South (S), North-East (NE), etc. Each region is a node (variable) in the constraint graph. The connections between regions form the edges (constraints). Each node is assigned a color such that adjacent nodes don’t have the same color. Only 3 colors (Red, Green, Blue) are used. The CSP solver must assign a valid color to each region ensuring constraint satisfaction. ✦ Summary Points Graph coloring is a type of CSP with finite discrete domains. It can be solved using: Backtracking Constraint propagation (e.g., forward checking, arc consistency) Heuristics (like minimum remaining values) It is used to model problems in: Map coloring Register allocation in compilers Scheduling problems Would you like a drawn graph or step-by-step solution to a real coloring problem as well?

Answer 9:

Here is the full **10-mark answer** to the question **“Explain with example graph coloring problem”**, exactly extracted from your provided textbook **Ai_unit_3.pdf**:

✅ Graph Coloring Problem (as explained in the book)

> **Graph coloring problem** is a type of **constraint satisfaction problem (CSP)**.
> A constraint graph consists of:

- **Nodes**: Variables (e.g., vertices of the graph that need to be colored)
- **Edges/Arcs**: Constraints (e.g., adjacent vertices must not have the same color)

✦ Formal Definition and Example (Map Coloring as Graph Coloring)

> **5.1.3.1 Map Colouring Problem**

1. The problem is to colour the regions of a given map such that no 2 adjacent regions have the same colour.

2. The regions in the map are the **variables**, and the set of possible colours for the regions is the **domain**.

3. The constraint is that:
   "**No two adjacent regions should have the same colour**."

Formal Representation:

- Variables: {N, NW, NE, M, MW, ME, S, SE}

- Domains: D = {Red, Green, Blue}

- Constraints: Adjacent regions must have different colors
  Example: N ≠ NW

We sometimes represent map colouring as a **graph coloring (constraint graph)** problem.
Typically, such a problem has many solutions.
The **topology of a constraint graph** can sometimes be used to identify solutions easily.

✦ Graph Coloring Example (with Figure Reference)

- In **Fig. 5.1.2**, a **map of regions** is shown, such as: North (N), Middle (M), South (S), North-East (NE), etc.

- Each region is a **node (variable)** in the constraint graph.

- The connections between regions form the **edges (constraints)**.

- Each node is assigned a color such that adjacent nodes don't have the same color.

- Only 3 colors (Red, Green, Blue) are used.

- The CSP solver must assign a valid color to each region ensuring constraint satisfaction.

✦ Summary Points

- Graph coloring is a type of CSP with finite discrete domains.
- It can be solved using:
    - Backtracking
    - Constraint propagation (e.g., forward checking, arc consistency)
    - Heuristics (like minimum remaining values)
- It is used to model problems in:
    - Map coloring
    - Register allocation in compilers
    - Scheduling problems

Would you like a drawn graph or step-by-step solution to a real coloring problem as well?

Question 10:

Here is the full 10-mark theoretical answer to the question: "How AI technique is used to solve tic-tac-toe problem", as exactly extracted from your provided textbook Ai_unit_3.pdf. ✅ Game Playing with Mini-Max – Tic-Tac-Toe Section 4.11.4 – Game Playing with Mini-Max-Tic-Tac-Toe (Noughts and Crosses) Assume that two players named MIN and MAX are playing the game. MAX is playing first. As you can see, initially MAX has nine possible moves. Play alternates between MAX and MIN until we reach leaf nodes corresponding to terminal states such that one player has 3 in a row or all the squares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX. High values are assumed to be good for MAX and bad for MIN. It is MAX's job to make use of the search tree to determine the best move. Static Evaluation Criteria: '+1' for a win, '0' for a draw, '-1' for a loss . ✅ Steps Demonstrated in Book Step 1: MAX's Turn MAX (playing 'X') considers all possible empty squares to make a move. Each move generates a new board configuration. Step 2: MIN's Turn MIN (playing 'O') evaluates each response to MAX's move. The tree is expanded with MIN's moves. For each resulting board state, a utility score is assigned based on the terminal outcome: win (+1), draw (0), loss (-1). Step 3: Backtracking Using Minimax At MIN nodes, the backed-up value is the minimum of children values. At MAX nodes, the backed-up value is the maximum of children values. ✅ Example of Tic-Tac-Toe Tree (as shown in Fig. 4.11.1 – 4.11.9) A search tree is created from the current configuration. The root node is the current board with MAX to move. Branches represent possible moves. Levels alternate between MAX and MIN. Leaf nodes are terminal states with assigned utilities. For instance: MAX move leads to: value = +1 (win) MIN move leads to: value = -1 (loss for MAX) Full board with no win: value = 0 (draw) ✅ Evaluation & Best Move Selection MAX uses the minimax decision by evaluating the maximum of minimum values of its subtrees. The AI system prunes unnecessary moves and selects only the optimal path to guarantee the best result against an optimal opponent . ✅ Key Advantages of Using AI for Tic-Tac-Toe Efficient Game Tree Search Using minimax, the entire game tree is explored up to terminal states. Perfect Strategy The AI never loses, as it evaluates all possible outcomes. Optimal Decision Making By simulating all future moves, AI determines the best current move. Static Evaluation Simple utility values allow rapid assessment of game states (+1, 0, -1). Let me know if you'd like a diagram of the game tree, flowchart of minimax working, or a simple Python implementation of AI vs. Human for tic-tac-toe.

Answer 10:

Here is the full **10-mark theoretical answer** to the question:
"How AI technique is used to solve tic-tac-toe problem",
as **exactly extracted from your provided textbook *Ai_unit_3.pdf***.

✅ Game Playing with Mini-Max – Tic-Tac-Toe

> **Section 4.11.4 – Game Playing with Mini-Max-Tic-Tac-Toe (Noughts and Crosses)**
>
> Assume that two players named **MIN and MAX** are playing the game.
> **MAX is playing first.**
>
> As you can see, initially MAX has nine possible moves.
>
> Play alternates between MAX and MIN until we reach leaf nodes corresponding to terminal states such that one player has 3 in a row or all the squares are filled.
>
> The number on each leaf node indicates the **utility value** of the terminal state from the point of view of **MAX**.
>
> High values are assumed to be **good for MAX** and **bad for MIN**.
>
> It is MAX's job to make use of the **search tree** to determine the **best move**.

Static Evaluation Criteria:
'+1' for a win,
'0' for a draw,
'-1' for a loss.

✅ Steps Demonstrated in Book

Step 1: MAX's Turn

MAX (playing 'X') considers all possible empty squares to make a move. Each move generates a new board configuration.

Step 2: MIN's Turn

MIN (playing 'O') evaluates each response to MAX's move. The tree is expanded with MIN's moves.
For each resulting board state, a **utility score** is assigned based on the terminal outcome: win (+1), draw (0), loss (-1).

Step 3: Backtracking Using Minimax

At **MIN nodes**, the backed-up value is the **minimum** of children values.
At **MAX nodes**, the backed-up value is the **maximum** of children values.

✅ Example of Tic-Tac-Toe Tree (as shown in Fig. 4.11.1 – 4.11.9)

A search tree is created from the current configuration.
The root node is the current board with MAX to move.
Branches represent possible moves.
Levels alternate between MAX and MIN.
Leaf nodes are terminal states with assigned utilities.

For instance:

- MAX move leads to: value = +1 (win)
- MIN move leads to: value = -1 (loss for MAX)

- Full board with no win: value = 0 (draw)

✅ Evaluation & Best Move Selection

MAX uses the minimax decision by evaluating the **maximum of minimum values** of its subtrees.
The AI system prunes unnecessary moves and selects only the **optimal path** to guarantee the best result against an optimal opponent.

✅ Key Advantages of Using AI for Tic-Tac-Toe

1. **Efficient Game Tree Search**
   - Using minimax, the entire game tree is explored up to terminal states.
2. **Perfect Strategy**
   - The AI never loses, as it evaluates all possible outcomes.
3. **Optimal Decision Making**
   - By simulating all future moves, AI determines the best current move.
4. **Static Evaluation**
   - Simple utility values allow rapid assessment of game states (+1, 0, -1).

Let me know if you'd like a diagram of the game tree, flowchart of minimax working, or a simple Python implementation of AI vs. Human for tic-tac-toe.

Question 11:

Here is the full 10-mark theoretical answer to the question: "List the inference rules used in propositional logic and explain them in detail with suitable examples", extracted from your provided book (Ai_unit_4.pdf) and expanded with additional clarity and examples where appropriate. ✅ Inference Rules in Propositional Logic (from the textbook) Inference rules are used to derive new conclusions from existing facts (propositions) in the knowledge base. These are essential for logical reasoning in AI systems. ✦ 1. Modus Ponens Symbolically: \alpha \Rightarrow \beta, \ \alpha \vdash \beta] Explanation: If we know that “if α then β” is true, and α is also true, we can infer that β must be true. Example (from book): (Signal Pole Ahead ∧ Signal Red) → Stop Signal Pole Ahead ∧ Signal Red ⟹ Stop ✦ 2. And-Elimination Symbolically: \alpha \land \beta \vdash \alpha \quad \text{or} \quad \beta] Explanation: If both α and β are true, then we can infer that either α is true or β is true individually. Example (from book): Given: (Signal Pole Ahead ∧ Signal Red) ⟹ Signal Red or Signal Pole Ahead ✦ 3. Unit Resolution Symbolically: (P \lor Q), \neg Q \vdash P] Explanation: From a disjunction (OR statement), and a negation of one part, we can infer the other. Example: P ∨ Q ¬Q ⟹ P ✦ 4. General Resolution Symbolically: (A \lor B), (\neg B \lor C) \vdash A \lor C] Explanation: Generalized form of unit resolution. When two clauses contain complementary literals (one positive, one negative), they cancel out, and the remaining parts form a new clause. Example: (P ∨ Q), (¬Q ∨ R) ⟹ (P ∨ R) This is a powerful rule forming the basis of automated theorem proving. It's complete for propositional logic when combined with search. ✦ 5. Conjunctive Normal Form (CNF) and Resolution To apply resolution, statements must be in Conjunctive Normal Form (CNF): a conjunction (AND) of clauses where each clause is a disjunction (OR) of literals. Steps to convert to CNF: Eliminate biconditional (↔) and implication (→) using equivalences. Move NOTs inwards using De Morgan’s laws. Distribute OR over AND. Example: (P → Q) becomes (¬P ∨ Q) (P ↔ Q) becomes (P → Q) ∧ (Q → P) → (¬P ∨ Q) ∧ (¬Q ∨ P) ✅ Additional Useful Inference Rules (beyond the textbook) To support deeper reasoning, here are some commonly used rules not listed in the book: Rule Description Example Double Negation ¬(¬P) ⇔ P ¬(¬Rain) ⇒ Rain De Morgan’s Laws ¬(P ∧ Q) ⇔ ¬P ∨ ¬Q ¬(Hot ∧ Dry) ⇒ ¬Hot ∨ ¬Dry Contrapositive P → Q ⇔ ¬Q → ¬P If it’s raining, streets are wet ⇔ If streets are not wet, it's not raining Hypothetical Syllogism (P → Q) ∧ (Q → R) ⇒ (P → R) If A ⇒ B and B ⇒ C, then A ⇒ C Disjunctive Syllogism P ∨ Q, ¬P ⇒ Q A or B, not A, therefore B ✅ Conclusion The inference rules in propositional logic form the foundation of deductive reasoning in AI systems. These rules: Ensure logical conclusions are valid Enable systems to automatically infer new facts Are used in knowledge-based systems, logical agents, and theorem provers These patterns of reasoning allow an AI agent to operate based on a knowledge base and make decisions even in partially observable or uncertain environments. Let me know if you'd like truth tables or a visual proof diagram for any of the rules above.

Answer 11:

Here is the **full 10-mark theoretical answer** to the question:
"List the inference rules used in propositional logic and explain them in detail with suitable examples",
extracted from your provided book (*Ai_unit_4.pdf*) and expanded with **additional clarity and examples** where appropriate.

✅ Inference Rules in Propositional Logic (from the textbook)

Inference rules are used to derive new conclusions from existing facts (propositions) in the knowledge base. These are essential for **logical reasoning in AI systems**.

✦ 1. Modus Ponens

Symbolically:

\alpha \Rightarrow \beta, \ \alpha \vdash \beta
]

Explanation:
If we know that “if α then β” is true, and α is also true, we can infer that β must be true.

Example (from book):

- (Signal Pole Ahead ∧ Signal Red) → Stop
- Signal Pole Ahead ∧ Signal Red
- ⇒ **Stop**

✦ 2. And-Elimination

Symbolically:

\alpha \land \beta \vdash \alpha \quad \text{or} \quad \beta
]

Explanation:
If both α and β are true, then we can infer that either α is true or β is true individually.

Example (from book):

- Given: (Signal Pole Ahead ∧ Signal Red)

- ⟹ **Signal Red** or **Signal Pole Ahead**

✦ 3. **Unit Resolution**

Symbolically:

(P \lor Q), \neg Q \vdash P
]

Explanation:
From a disjunction (OR statement), and a negation of one part, we can infer the other.

Example:

- P ∨ Q
- ¬Q
- ⟹ **P**

✦ 4. **General Resolution**

Symbolically:

(A \lor B), (\neg B \lor C) \vdash A \lor C
]

Explanation:
Generalized form of unit resolution. When two clauses contain complementary literals (one positive, one negative), they cancel out, and the remaining parts form a new clause.

Example:

- (P ∨ Q), (¬Q ∨ R)

- ⟹ (P ∨ R)

This is a powerful rule forming the **basis of automated theorem proving**. It's **complete** for propositional logic when combined with **search**.

✦ 5. Conjunctive Normal Form (CNF) and Resolution

To apply resolution, statements must be in **Conjunctive Normal Form** (CNF): a conjunction (AND) of clauses where each clause is a disjunction (OR) of literals.

Steps to convert to CNF:

1. Eliminate biconditional (↔) and implication (→) using equivalences.
2. Move NOTs inwards using De Morgan's laws.
3. Distribute OR over AND.

Example:

- (P → Q) becomes (¬P ∨ Q)
- (P ↔ Q) becomes (P → Q) ∧ (Q → P) → (¬P ∨ Q) ∧ (¬Q ∨ P)

✅ Additional Useful Inference Rules (beyond the textbook)

To support deeper reasoning, here are some commonly used rules not listed in the book:

Rule	Description	Example
Double Negation	¬(¬P) ⇔ P	¬(¬Rain) ⇒ Rain
De Morgan's Laws	¬(P ∧ Q) ⇔ ¬P ∨ ¬Q	¬(Hot ∧ Dry) ⇒ ¬Hot ∨ ¬Dry
Contrapositive	P → Q ⇔ ¬Q → ¬P	If it's raining, streets are wet ⇔ If streets are not wet, it's not raining
Hypothetical Syllogism	(P → Q) ∧ (Q → R) ⇒ (P → R)	If A ⇒ B and B ⇒ C, then A ⇒ C
Disjunctive Syllogism	P ∨ Q, ¬P ⇒ Q	A or B, not A, therefore B

✅ Conclusion

The inference rules in propositional logic form the foundation of **deductive reasoning** in AI systems. These rules:

- Ensure logical conclusions are **valid**
- Enable systems to **automatically infer** new facts
- Are used in **knowledge-based systems**, **logical agents**, and **theorem provers**

These patterns of reasoning allow an AI agent to operate based on a **knowledge base** and make decisions even in **partially observable or uncertain environments**.

Let me know if you'd like truth tables or a visual proof diagram for any of the rules above.

Question 12:

Here is the detailed 50-mark theoretical answer on: ✅ Syntax and Semantics of First Order Logic (FOL) This includes exact extracts from your textbook (Ai_unit_4.pdf) with added detailed explanations and examples for clarity. 📘 1. Introduction to First Order Logic (FOL) First Order Logic (FOL) is an expressive language that has well-defined syntax and semantics. It uses: Objects: e.g. people, numbers, houses. Relations: e.g. brother_of, bigger_than, owns. Functions: e.g. father_of, square_of. Example: “Two plus five equals seven” Objects: Two, Five, Seven Relation: equals Function: plus ✍️ 2. Syntax of First Order Logic Syntax defines the legal sentences of a logic language. It determines how symbols can be combined. 🔹 Components of Syntax Constants: Represent objects. E.g.: Ram, Cinderella, Red Variables: Denote unspecified objects. E.g.: X, Y, Z Predicates: Express relations between objects. E.g.: Likes(X, Y), Brother(Ram, Shyam) Functions: Map objects to objects. E.g.: FatherOf(X), RightLegShoe(X) Connectives: ∧ (and), ∨ (or), ¬ (not), ⇒ (implies), ⇔ (if and only if) Quantifiers: ∀ (for all), ∃ (there exists) 🔸 Sentences in FOL Atomic Sentence: Predicate with terms. Example: Loves(Ram, Sita) Complex Sentence: Built using connectives and quantifiers. Example: ∀x (Human(x) ⇒ Mortal(x)) 🖋️ Example (from textbook) "Multitalented King Krishna Ruled Dwarika" Objects: Krishna, Dwarika Relation: Ruled Functions: King, Multitalented 📚 3. Semantics of First Order Logic Semantics tells us what statements mean—how we interpret the syntax in the context of a model. 🔹 Interpretation (from textbook) An interpretation assigns meaning to each symbol: A domain D D: the set of all objects. Constants → elements of D D Variables → values in D D Functions → maps $D n \rightarrow D$ D n →D Predicates → map $D n \rightarrow \{True, False\}$ D n →{True,False} Given an interpretation: The truth value of an atomic sentence is determined by the predicate's value under interpretation 🔸 Truth Assignment (Evaluation) ¬P is true if P is false. P ∧ Q is true if both are true. P ∨ Q is true if either is true. P ⇒ Q is true unless P is true and Q is false. P ⇔ Q is true if both have the same truth value. Quantifiers: ∀x P(x) is true if P(x) is true for all x. ∃x P(x) is true if P(x) is true for at least one x. 🔁 4. Quantifiers in FOL 🔸 Universal Quantifier (∀) Symbol: ∀ Meaning: "For all" Example: "All humans are mortal" → ∀x (Human(x) ⇒ Mortal(x)) "All princesses are people" → ∀x (Princess(x) ⇒ Person(x)) 🔸 Existential Quantifier (∃) Symbol: ∃ Meaning: "There exists" Example: "There exists someone who likes cricket" → ∃x (Person(x) ∧ Likes(x, Cricket)) 🔁 Properties of Quantifiers Quantifiers of same type commute: ∀x ∀y ≡ ∀y ∀x ∃x ∃y ≡ ∃y ∃x Quantifiers of different type do not commute: ∃x ∀y Loves(x, y) ≠ ∀y ∃x Loves(x, y) First means "someone loves everyone", second means "everyone is loved by someone" ✅ 5. Extended Examples 🧠 Example 1: "Brothers are siblings" ∀x, y (Brother(x, y) ⇒ Sibling(x, y)) 🧠 Example 2: "No one likes taxes" ¬∃x Likes(x, Taxes) ≡ ∀x ¬Likes(x, Taxes) 🧠 Example 3: Blocks World (robot planning) On(c, a) OnTable(a) Clear(c) HandEmpty 📐 6. Differences Between Syntax and Semantics Feature Syntax Semantics Definition Structure / grammar of expressions Meaning / interpretation of expressions Validation Checks if sentence is well-formed Checks if sentence is true in a model Example ∀x (P(x) ∧ Q(x)) is syntactically valid Whether it's true depends on model & domain 🌟 7. Why FOL is Powerful Can express facts, rules, generalizations More expressive than propositional logic Basis for AI knowledge bases, expert systems, and automated reasoning 📝 8. Summary Syntax defines the valid sentence structure in FOL using symbols, terms, predicates, functions, and quantifiers. Semantics explains how these sentences are interpreted and how truth values are derived under a given model. FOL enables reasoning about objects, their properties, and relationships, making it a core tool for Artificial Intelligence. Let me know if you'd like: A printable version of this Diagrams showing syntax trees FOL vs Propositional logic comparison table

Answer 12:

Here is the **detailed 50-mark theoretical answer** on:

✅ Syntax and Semantics of First Order Logic (FOL)

This includes **exact extracts from your textbook** (*Ai_unit_4.pdf*) with added detailed explanations and examples for clarity.

📘 1. Introduction to First Order Logic (FOL)

> First Order Logic (FOL) is an expressive language that has well-defined **syntax** and **semantics**.

It uses:

- **Objects**: e.g. people, numbers, houses.
- **Relations**: e.g. brother_of, bigger_than, owns.
- **Functions**: e.g. father_of, square_of.

Example:

- "Two plus five equals seven"
  - **Objects**: Two, Five, Seven
  - **Relation**: equals
  - **Function**: plus

✍️ 2. Syntax of First Order Logic

Syntax defines the legal sentences of a logic language. It determines how symbols can be combined.

🔹 Components of Syntax

1. **Constants**: Represent objects.
   E.g.: `Ram`, `Cinderella`, `Red`
2. **Variables**: Denote unspecified objects.
   E.g.: `X`, `Y`, `Z`
3. **Predicates**: Express relations between objects.
   E.g.: `Likes(X, Y)`, `Brother(Ram, Shyam)`
4. **Functions**: Map objects to objects.
   E.g.: `FatherOf(X)`, `RightLegShoe(X)`

5. **Connectives**:
   - ∧ (and), ∨ (or), ¬ (not), ⇒ (implies), ⇔ (if and only if)
6. **Quantifiers**:
   - ∀ (for all), ∃ (there exists)

🔸 Sentences in FOL

- **Atomic Sentence**: Predicate with terms.
  Example: `Loves(Ram, Sita)`
- **Complex Sentence**: Built using connectives and quantifiers.
  Example: `∀x (Human(x) ⇒ Mortal(x))`

🖋 Example (from textbook)

"Multitalented King Krishna Ruled Dwarika"

- Objects: Krishna, Dwarika
- Relation: Ruled
- Functions: King, Multitalented

📚 3. Semantics of First Order Logic

Semantics tells us what statements mean—how we interpret the syntax in the context of a model.

🔹 Interpretation (from textbook)

An **interpretation** assigns meaning to each symbol:

- A **domain** DD: the set of all objects.

- Constants → elements of DD
- Variables → values in DD
- Functions → maps $D^n \to D$Dn → D
- Predicates → map $D^n \to \{True, False\}$Dn → {True, False}

Given an interpretation:

- The **truth value** of an atomic sentence is determined by the predicate's value under interpretation

---

Truth Assignment (Evaluation)

- `¬P` is true if `P` is false.
- `P ∧ Q` is true if both are true.
- `P ∨ Q` is true if either is true.
- `P ⇒ Q` is true unless P is true and Q is false.
- `P ⇔ Q` is true if both have the same truth value.
- Quantifiers:
    - `∀x P(x)` is true if P(x) is true for all x.
    - `∃x P(x)` is true if P(x) is true for at least one x.

---

4. Quantifiers in FOL

Universal Quantifier (∀)

- Symbol: ∀
- Meaning: "For all"
- Example:
    - “All humans are mortal” → `∀x (Human(x) ⇒ Mortal(x))`
    - “All princesses are people” → `∀x (Princess(x) ⇒ Person(x))`

🔸 Existential Quantifier (∃)

- Symbol: ∃
- Meaning: "There exists"
- Example:
    - “There exists someone who likes cricket” → `∃x (Person(x) ∧ Likes(x, Cricket))`

🔁 Properties of Quantifiers

- Quantifiers of same type **commute**:
    - $\forall x\ \forall y \equiv \forall y\ \forall x$
    - $\exists x\ \exists y \equiv \exists y\ \exists x$
- Quantifiers of **different type do not commute**:
    - `∃x ∀y Loves(x, y)` ≠ `∀y ∃x Loves(x, y)`
    - First means "someone loves everyone", second means "everyone is loved by someone"

✅ 5. Extended Examples

🧠 Example 1: “Brothers are siblings”

`∀x, y (Brother(x, y) ⇒ Sibling(x, y))`

🧠 Example 2: “No one likes taxes”

`¬∃x Likes(x, Taxes)` ≡ `∀x ¬Likes(x, Taxes)`

🧠 Example 3: Blocks World (robot planning)

- `On(c, a)`
- `OnTable(a)`
- `Clear(c)`
- `HandEmpty`

📐 6. Differences Between Syntax and Semantics

Feature	Syntax	Semantics
Definition	Structure / grammar of expressions	Meaning / interpretation of expressions
Validation	Checks if sentence is well-formed	Checks if sentence is true in a model
Example	`∀x (P(x) ∧ Q(x))` is syntactically valid	Whether it’s true depends on model & domain

---

🌟 7. Why FOL is Powerful

- Can express **facts, rules, generalizations**
- More expressive than propositional logic
- Basis for **AI knowledge bases**, **expert systems**, and **automated reasoning**

📝 8. Summary

- **Syntax** defines the valid sentence structure in FOL using symbols, terms, predicates, functions, and quantifiers.
- **Semantics** explains how these sentences are **interpreted** and how **truth values** are derived under a given model.
- FOL enables **reasoning** about objects, their **properties**, and **relationships**, making it a core tool for **Artificial Intelligence**.

Let me know if you'd like:

- A printable version of this
- Diagrams showing syntax trees
- FOL vs Propositional logic comparison table

Question 13:

Here is a comprehensive 30-mark answer to the question: "Detail the algorithm for deciding entailment in propositional logic", combining exact text from your provided textbook (Ai_unit_4.pdf) and detailed elaboration to enhance understanding. ✅ What Is Entailment? Entailment in logic means determining whether a knowledge base (KB) logically implies a sentence (α). This is written as: KB \models \alpha] It means α is true in all models where KB is true. ✅ Algorithm for Deciding Entailment: Truth Table Enumeration The truth-table enumeration algorithm is a sound and complete method for checking propositional entailment. ✦ Purpose: To determine if $KB \models \alpha$ KB⊨α by checking all possible models (assignments of truth values) to the variables. 📌 TT-Entails Algorithm (from book) Function: text Copy TT-Entails(KB, α) Inputs: KB (knowledge base), α (proposition), both in propositional logic Returns: true if α is entailed by KB, else false Steps: Generate a list of all propositional symbols from KB and α. Enumerate all possible truth value assignments (models). Check for each model: If KB is true in the model Then check whether α is also true in that model If α is true in all models where KB is true, return true. Otherwise, return false. 🧠 Internal Subroutine: TT-CHECK-ALL text Copy Function TT-CHECK-ALL(KB, α, symbols, model) if symbols is empty: if PL-TRUE(KB, model) then return PL-TRUE(α, model) else return true else: P ← first symbol rest ← remaining symbols return (TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))) Explanation: PL-TRUE checks if a sentence is true in the given model. EXTEND(P, value, model) adds an assignment P=value to the model. 📘 Extracted from Book: "This algorithm performs a recursive enumeration of a finite space of assignments to variables. It is sound and complete because it directly implements the definition of entailment" . ✅

Complexity Analysis Measure Value Time Complexity $O(2^n)$ O(2 n) — exponential Space Complexity $O(n)$ O(n) — for recursion depth (symbols) ✅ Properties Sound: If it returns true, the entailment is correct. Complete: It explores all possible models (assignments). CO-NP-complete: Propositional entailment is computationally hard . ✅ Alternative Approaches (from the book) ✦ 1. Resolution Algorithm Converts KB ∧ ¬α to CNF. Repeatedly applies the resolution rule to derive new clauses. If an empty clause is derived, KB ⊨ α. Refutation complete: always gives a result . ✦ 2. DPLL Algorithm (Davis–Putnam–Logemann–Loveland) Improved backtracking search. Uses early termination, pure symbol, and unit clause heuristics. Much more efficient than TT-Entails in practice . ✦ 3. WALKSAT Algorithm A local search algorithm for satisfiability. Fast, but incomplete — may fail to detect unsatisfiability . ✅ Example Let: KB = {P → Q, P} α = Q Using TT-Entails: Possible assignments: (P=T, Q=T), (P=T, Q=F), (P=F, Q=T), (P=F, Q=F) In models where KB is true: Only (P=T, Q=T) and (P=F, Q=anything) In all such models, Q is true ⇒ KB ⊨ Q ✅ ✅ Conclusion The Truth Table Entailment Algorithm is the foundational method for propositional logic entailment. Though simple, it is computationally expensive and not scalable for large KBs. Optimized algorithms like Resolution, DPLL, and WALKSAT offer better performance in real-world AI applications. Let me know if you'd like diagrams or pseudocode formats for your notes or viva prep!

Answer 13:

Here is a comprehensive **30-mark answer** to the question:
"Detail the algorithm for deciding entailment in propositional logic", combining **exact text from your provided textbook (Ai_unit_4.pdf)** and detailed elaboration to enhance understanding.

✅ What Is Entailment?

> **Entailment** in logic means determining whether a knowledge base (KB) **logically implies** a sentence (α).
> This is written as:

KB \models \alpha
]

> It means **α is true in all models where KB is true**.

✅ Algorithm for Deciding Entailment: **Truth Table Enumeration**

The **truth-table enumeration algorithm** is a sound and complete method for checking propositional entailment.

✦ Purpose:

To determine if $KB \vDash \alpha$KB ⊨ α by checking all possible models (assignments of truth values) to the variables.

📌 TT-Entails Algorithm (from book)

Function:

```text
TT-Entails(KB, α)
  Inputs: KB (knowledge base), α (proposition), both in propositional logic
  Returns: true if α is entailed by KB, else false
```

Steps:

1. Generate a list of all **propositional symbols** from KB and α.
2. Enumerate all **possible truth value assignments** (models).
3. Check for **each model**:
   - If KB is true in the model
   - Then check whether α is also true in that model
4. If α is true in **all models** where KB is true, return **true**.
   Otherwise, return **false**.

🧠 Internal Subroutine: `TT-CHECK-ALL`

```text
Function TT-CHECK-ALL(KB, α, symbols, model)
  if symbols is empty:
```

```
    if PL-TRUE(KB, model) then return PL-TRUE(α, model)
    else return true
else:
    P ← first symbol
    rest ← remaining symbols
    return (TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model))
        and TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model)))
```

Explanation:

- `PL-TRUE` checks if a sentence is true in the given model.
- `EXTEND(P, value, model)` adds an assignment `P=value` to the model.

📘 **Extracted from Book**:

> "This algorithm performs a recursive enumeration of a finite space of assignments to variables. It is sound and complete because it directly implements the definition of entailment".

✅ Complexity Analysis

Measure	Value
Time Complexity	$O(2^n)$O(2n) — exponential
Space Complexity	$O(n)$O(n) — for recursion depth (symbols)

✅ Properties

- **Sound**: If it returns true, the entailment is correct.
- **Complete**: It explores all possible models (assignments).
- **CO-NP-complete**: Propositional entailment is computationally hard.

✅ Alternative Approaches (from the book)

✦ 1. Resolution Algorithm

- Converts KB ∧ ¬α to CNF.
- Repeatedly applies the **resolution rule** to derive new clauses.
- If an **empty clause** is derived, KB ⊨ α.
- **Refutation complete**: always gives a result.

✦ 2. DPLL Algorithm (Davis–Putnam–Logemann–Loveland)

- Improved backtracking search.
- Uses **early termination**, **pure symbol**, and **unit clause heuristics**.
- Much more efficient than TT-Entails in practice.

✦ 3. WALKSAT Algorithm

- A **local search** algorithm for satisfiability.
- Fast, but **incomplete** — may fail to detect unsatisfiability.

✅ Example

Let:

- KB = {P → Q, P}
- α = Q

Using TT-Entails:

- Possible assignments:
  - (P=T, Q=T), (P=T, Q=F), (P=F, Q=T), (P=F, Q=F)

- In models where KB is true:
  - Only (P=T, Q=T) and (P=F, Q=anything)
- In all such models, Q is true ⇒ **KB ⊨ Q** ✅

✅ Conclusion

- The **Truth Table Entailment Algorithm** is the foundational method for propositional logic entailment.
- Though simple, it is computationally expensive and not scalable for large KBs.
- **Optimized algorithms** like **Resolution**, **DPLL**, and **WALKSAT** offer better performance in real-world AI applications.

Let me know if you'd like diagrams or pseudocode formats for your notes or viva prep!
