(3/12/24)

LAB-09:

Forward Chaining:

Problem Statement:

As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen.

• Solution:

1. It is a crime for an American to sell weapons to hostile nations

$American(p) \wedge Weapon(q) \wedge Sells(p, q, r) \wedge Hostile(r) \Rightarrow Criminal(p)$

p: person, q: thing, r: sells to r

p, q, r are variables

2. Country A has some missiles

$\exists x\ (Owns(A, x) \wedge Missile(x))$

3. All of missiles were sold to country A by Robert

$\forall x (Missile(x) \wedge Owns(A, x) \Rightarrow Sells(Robert, x, A))$

4. Missles are weapons

$Missile(x) \Rightarrow Weapon(x)$

5. Enemy of America is known as hostile

$\forall x\ (Enemy(x, America) \Rightarrow Hostile(x))$

6. Robert is an American

$American(Robert)$

7. The country A, an enemy of America

$Enemy(A, America)$

• Observation:

→ From (5) & (7), we can say that A is enemy of America and hence it is a hostile of America ($Hostile(A)$)

→ From (2) & (4), country A has some missiles & missile are weapons, so country A has weapons ($Weapons(x)$)

→ From ③ & ④ , Missile(x) ∧ Owns(A, x) ⇒ Sells(Robert, x, A).

All missiles, which are weapons, were sold to country A by Robert.

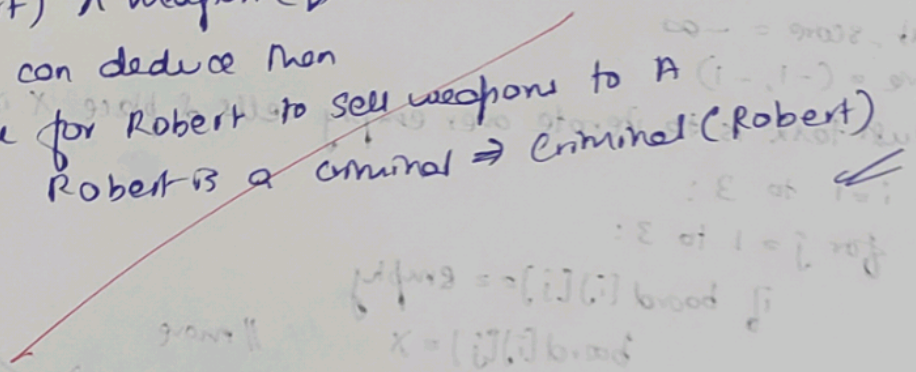
→ Considering statement ①,

American(Robert) ∧ weapon(q) ∧ sells(Robert, x, A) ∧ Hostile(A)

from this, we can deduce that

It is a crime for Robert to sell weapons to A

So Robert is a criminal ⇒ Criminal(Robert)



Output:

Robert is a criminal

LAB-10 (3/12/24)

Min-Max for TicTacToe:

AI - X
Human - O

X O X
O O X
_ _ _

Algorithm:

```
def find_best_move(board):
    best_score = -∞
    move = (-1, -1)
    #our task is to iterate over empty cells & place X in the cell
    for i=1 to 3:
        for j=1 to 3:
            if board[i][j] == empty
                board[i][j] = X          // move
                score = minmax(board, 0, False)   → we're trying if we can maximize AI to win or minimize Human to win
                board[i][j] = empty      //undo

                if score > best_score:
                    best_score = score
                    move = (i, j)

    return move

if is_maximizing = False
    ai
    human to
    ~~minimax~~

def minmax(board, depth, is_maximizing):
    if check(board, AI): return 10 - depth    // prioritizing win
    if check(board, HUMAN): return depth - 10  // delaying loss

    if is_maximizing:
        best_score = -∞
        for i=1 to 3:
            for j=1 to 3:
                if board[i][j] == empty:
                    board[i][j] = AI          // move
                    score = minmax(board, depth+1, False)
                    board[i][j] = empty       //undo
                    best_score = max(best_score, score)
        return best_score

    else:
        True
        min
        return best_score
```

output:

Current Board:

X O X

O O X

– – –

The best move for AI is (2, 2)

α-β pruning for 8-queens problem:

Algorithm:

```
def alpha_beta(self, board, col, alpha, beta, maximizing_player):
    if col >= self.size:                                  → valid soln is found.
        return 0, [row[:] for row in board]

    if maximizing_player:                  # maximizes score
        max_eval = -∞                                     → iterates through each col & if placing a queen is safe.
        best_board = None
        for row in range(self.size):
            if self.is_safe(board, row, col):
                board[row][col] = 1
                eval_score, pot_board = alpha_beta(board, col+1, α, β, False)
                board[row][col] = 0
                if eval_score > max:
                    max = eval_score
                    best_board = pot_board
                    alpha = max(alpha, eval_score)
                if beta <= alpha:          # Beta cutoff
                    break

        return max_eval, best_board

    else:                                  # minimizing score
        True
    beta = min( )
        ~~beta~~
        beta <= alpha:          # alpha cutoff
            break
        return min_eval, best_board
```

Proceed

Output:

Solution found