3/12/24

Forward chaining.

Ex:
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.

Prove - Robert is criminal.

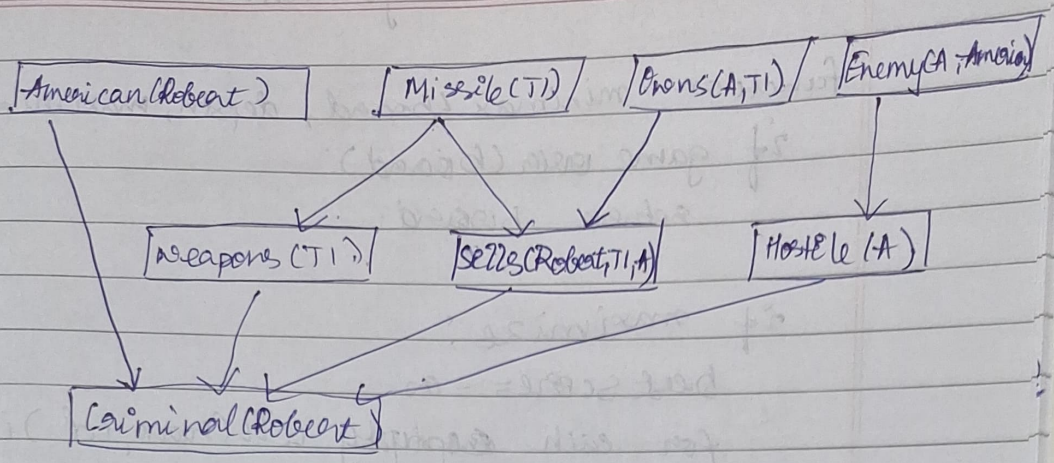
Its a crime for an American to sell weapons to hostile nations.

American(x) ∧ weapons(y) ∧ sells(x, y, z) ∧ Hostile(z)

Country A has some missiles.
∃x Owns(A, x) ∧ Missile(x).
Robert is American American(Robert)
All the missiles were sold to country A by Robert
∀x Missile(x) ∧ Owns(A, x) ⇒ sells(Robert, x, A).

Missiles are weapons
Missile(x) ⇒ weapon(x)
Country A, enemy of America Enemy(A, America)
Enemy of America is known as hostile.
∀x Enemy(x, America) ⇒ Hostile(x)

To prove criminal(Robert)



American(x) ∧ weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z)
⇒ Criminal(x)

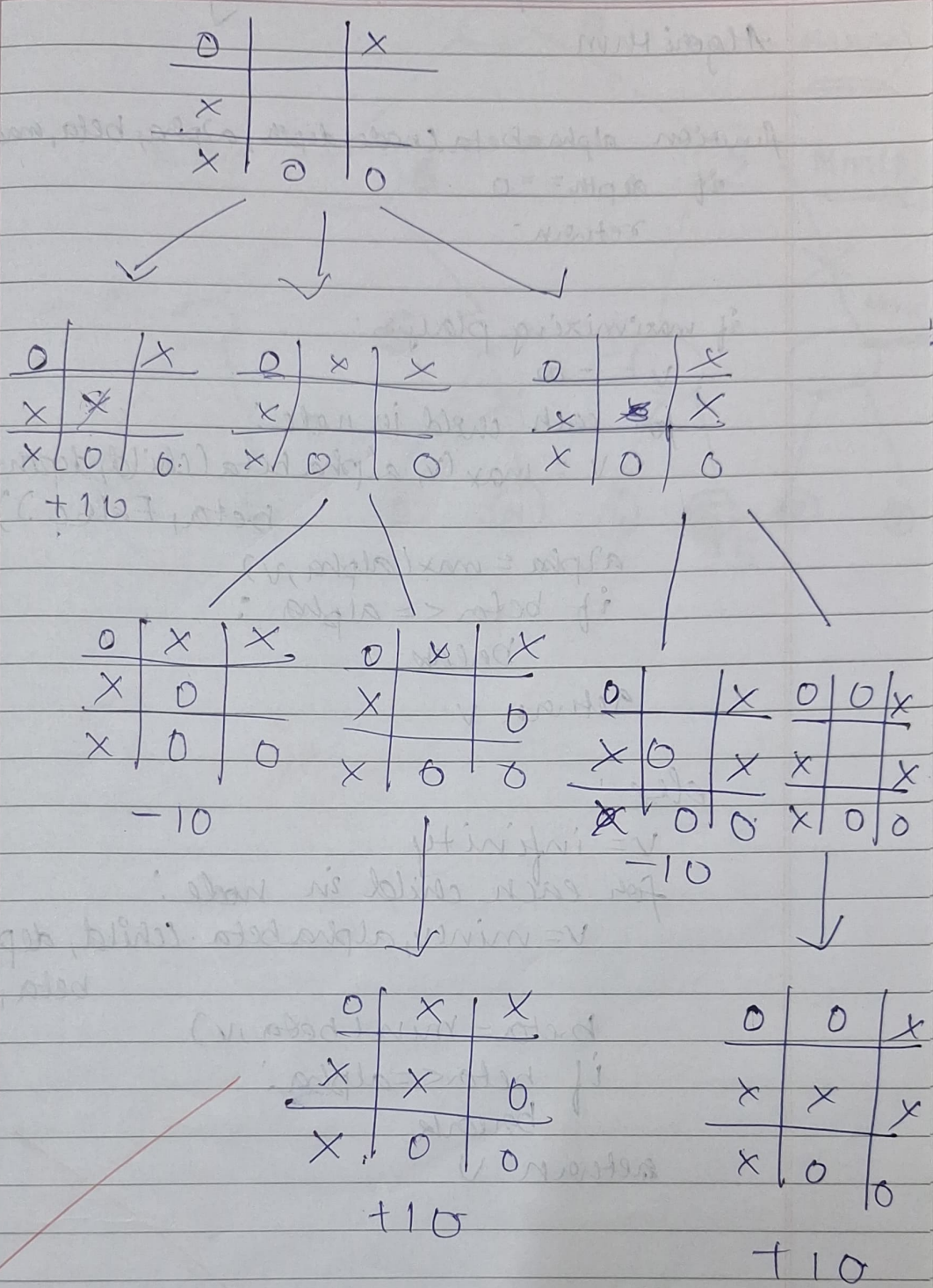
¬Criminal(x)

MinMax Algorithm.

```
function minimax (board, depth, maximize):
  if game over (board):
    return board

  if maximize:
    best_score = -∞
    for each empty cell (row, col) in board:
      simulate move (board, row, col, 'X')
      score = minimax (board, depth+1, False)
      undo_move (board, row, col)
      best_score = max (best_score, score)
    return best_score
  else:
    best_score = +∞
    for each empty cell (row, col) in board:
      simulate_move (board, row, col, 'O')
      score = minimax (board, depth+1, True)
      undo_move (board, row, col)
      best_score = min (best_score, score)
  return best_score.

function find_best_move (board, maximize):
  best_score = -∞ if maximize else +∞
  for each empty cell (row, col) in board:
    simulate_move (board, row, col, 'X' if else 'O')
    move_score = minimax (board, 0, not maximize)
    undo_move (board, row, col)
    if isMaximize and move_score > best_score:
      best_score = move_score
      best_move = (row, col)
    elif not maximize and move_score < best_score:
      best_score = move_score
      best_move = (row, col)
return best_move
```



Max
Max

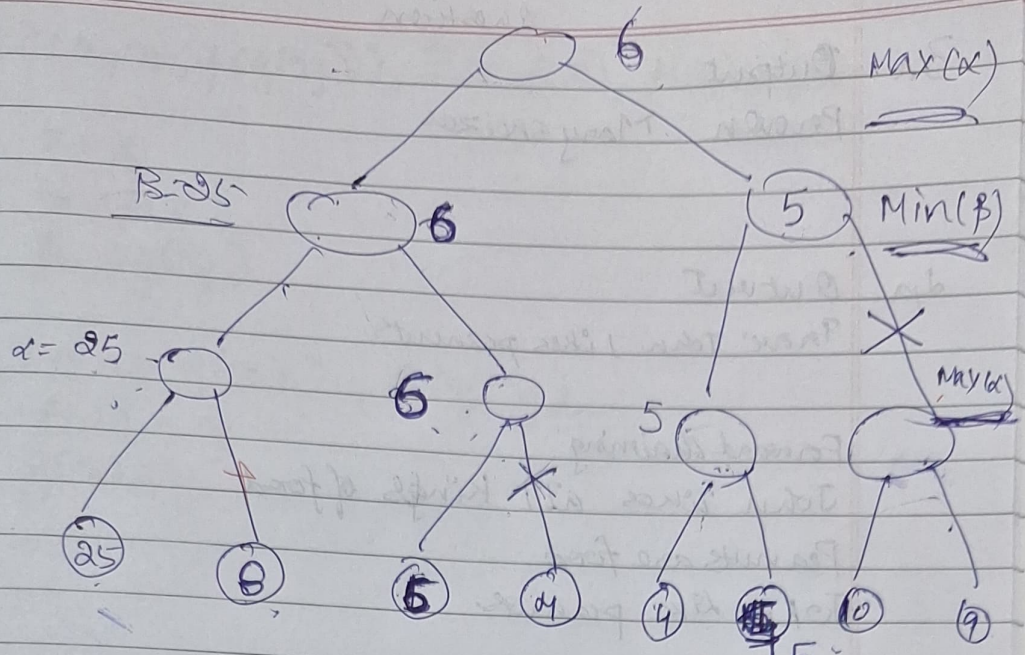
Alpha Beta Pruning

Algorithm

```
function alpha-beta(node, depth, alpha, beta, maximizing_player):
    if depth == 0
        return

    if maximizing_player:
        v = -∞
        for each child in node:
            v = max(v, alphabeta(child, depth-1, alpha,
                                 beta, False))
            alpha = max(alpha, v)
            if beta <= alpha:
                break
        return v

    else:
        v = infinity
        for each child in node:
            v = min(v, alphabeta(child, depth-1, alpha,
                                 beta, True))
            beta = min(beta, v)
            if beta <= alpha:
                break
        return v
```



3/12/20