week 10 :

Resolution in First-order Logic

Date __/__/__
Page

Step- 1 :-

Express Knowledge Base (KB) in first-order logic

1. Premises :
   - $\forall x$ Food(x) $\rightarrow$ Likes(John, x)
     (John likes all kind of food)
   - Food(Apple) $\wedge$ Food(vegetables)
     (Apple and vegetables are food)
   - $\forall x \forall y$ (Eats(y, x) $\wedge \neg$ Killed(y)) $\rightarrow$ Food(x)
     (Anything anyone eats & is not killed is food)
   - Eats(Anil, Peanuts) $\wedge \neg$ Killed(Anil)
     (Anil eats peanuts and is still alive)
   - $\forall x$ Eats(Anil, x) $\rightarrow$ Eats(Harry, x)
     (Harry eats everything that Anil eats)
   - $\forall x$ Alive(x) $\rightarrow \neg$ Killed(x)
     (Anyone who is alive is not killed)
   - $\forall x \neg$ Killed(x) $\rightarrow$ Alive(x)
     (Anyone who is not killed is alive)

2. Goal(n)
   - Proves : Likes(John, Peanuts)

Step 2 : Convert to Clausal Form

Convert each statement into conjunctive Normal Form (CNF)

1. $\neg$ Food(x) V Likes(John, x)
2. Food(Apple)
   Food(Vegetables)
3. $\neg$ Eats(y, x) V Killed(y) V Food(x)
4. Eats(Anil, Peanuts) $\neg$ Killed(Anil)
5. $\neg$ Eats(Anil, x) V Eats(Harry, x)
6. $\neg$ Alive(x) V $\neg$ Killed(x)
7. Killed(x) V Alive(x)

Goal : $\neg$ Likes(John, Peanuts)

Date __/__/__
Page

Step 3 : Apply Resolution

(1) From (4) :
Eats (Anil, Peanuts) ∧ ¬Killed (Anil)

(2) From (3) :
Substitute y = Anil, x = Peanuts :
¬Eats (Anil, Peanuts) ∨ Killed (Anil) ∨ Food (Peanuts)

Resolve with Eats (Anil, Peanuts) :
Killed (Anil) ∨ Food (Peanuts)
Resolve with ¬Killed (Anil) :
Food (Peanuts)

(3) From (1) :
Substitute x = Peanuts :
¬Food (Peanuts) ∨ Likes (John, Peanuts)
Resolve with Food (Peanuts) :
Likes (John, Peanuts)

(4) Negation of goal ¬Likes (John, Peanuts) is resolved

→ Output
By resolution, Likes (John, Peanuts) is proven.

Date / /
Page

Given the Knowledge Base

a. John likes all kind of food
b. Apple and vegetables are food
c. Anything any one eats & not killed is food
d. Anil eats Peanuts and still Alive.
e. Harry eats everything that Anil eats
f. Anyone who is alive implies not killed
g. Anyone who is not killed implies alive.

Step 1:-
Expressing KB in FOL

a. $\forall x : food(x) \rightarrow likes(John, x)$
b. $food(Apple) \wedge food(vegetables)$
c. $\forall x \forall y : eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$
d. $eats(Anil, Peanuts) \wedge alive(Anil)$
e. $\forall x : eats(Anil, x) \rightarrow eats(Harry, x)$
f. $\forall x : \neg killed(x) \rightarrow alive(x)$
g. $\forall x : alive(x) \rightarrow \neg killed(x)$
h. $likes(John, Peanuts)$.

Step 2:-
converting Clausal Form

a. $\neg food(x) \vee likes(John, x)$
b. $food(Apple)$
c. $food(vegetables)$
d. $\neg eats(y, z) \vee killed(y) \vee food(z)$
e. $eats(Anil, Peanuts)$

Date __/__/__
Page

f. alive(Anil)
g. ¬ eats(Anil, w) ∨ eats(Harry, w)
h. killed(g) ∨ alive(g)
i. ¬ alive(k) ∨ ¬ killed(k)
j. likes(John, Peanuts)

Proof by Resolution

① From (g):
eats(Anil, Peanuts)

② From (a):
¬ eats(y, z) ∨ killed(y) ∨ Food(z)
Put y = Anil & z = Peanuts
¬ eats(Anil, Peanuts) ∨ killed(Anil) ∨ Food(Peanuts)

③ From (i):
¬ food(x) ∨ likes(John, x)
Put x = Peanuts

¬ food(x) ∨ likes(John, Peanuts)

Out puts
Hence
likes(John, Peanuts)
John likes Peanuts.

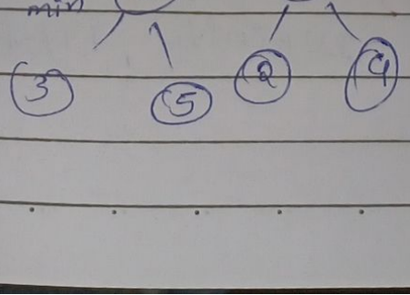
Min MAX Algorithm

Date __/__/__
Page _____

Algorithm codes:-

```
import math

def minmax(curDepth, nodeIndex, maxTurn
        score, targetDepth):
    if (curDepth == targetDepth):
        return score[nodeIndex]

    if (maxTurn):
        return max(minimax(curDepth+1, nodeIndex
                *2, False, score, targetDepth),
            minimax(curDepth+1, nodeIndex*2+1,
                False, score, targetDepth))
    else:
        return min(minimax(currentDepth+1, nodeIndex
                *2, True, score, targetDepth),
            minimax(curDepth+1, nodeIndex*2+1,
                True, score, targetDepth))

score = [3, 5, 2, 9, 12, 5, 23, 28]

treeDepth = math.log(len(score), 2)
print("The optimal value is", end="")

print(minmax(0, 0, true, score, treeDepth))
```

→



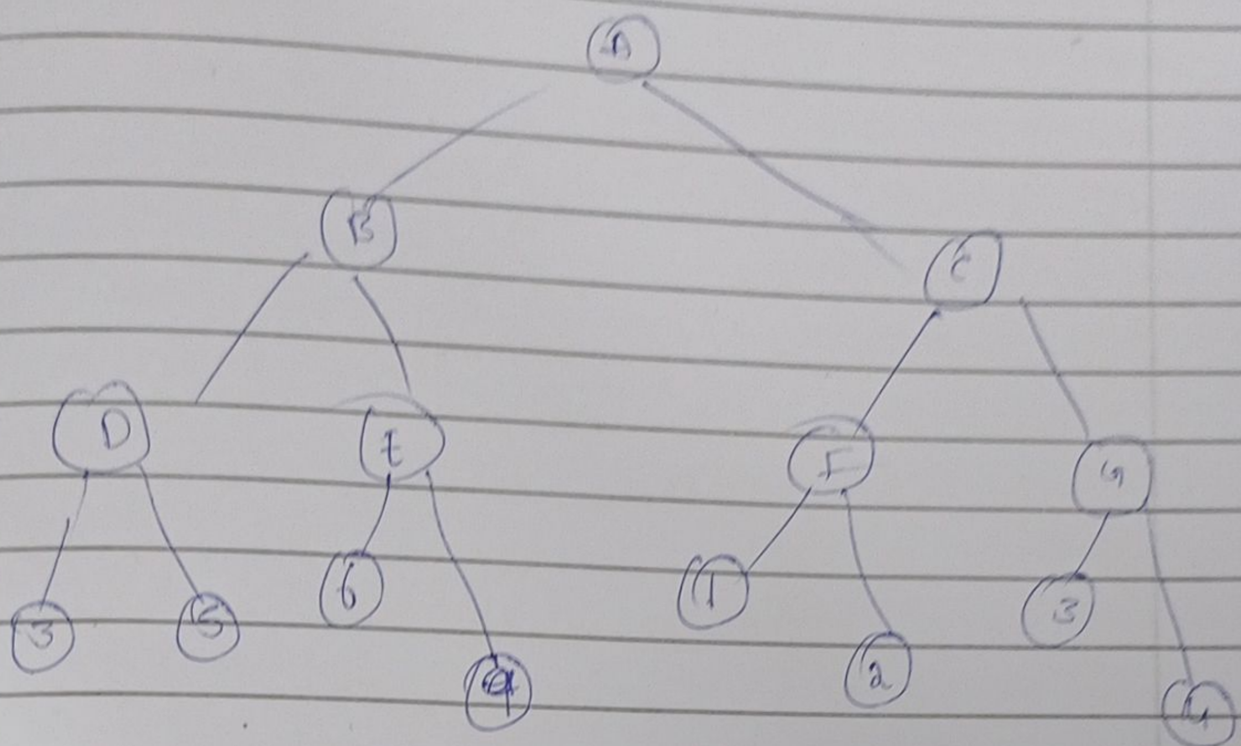
Date __/__/__
Page _____

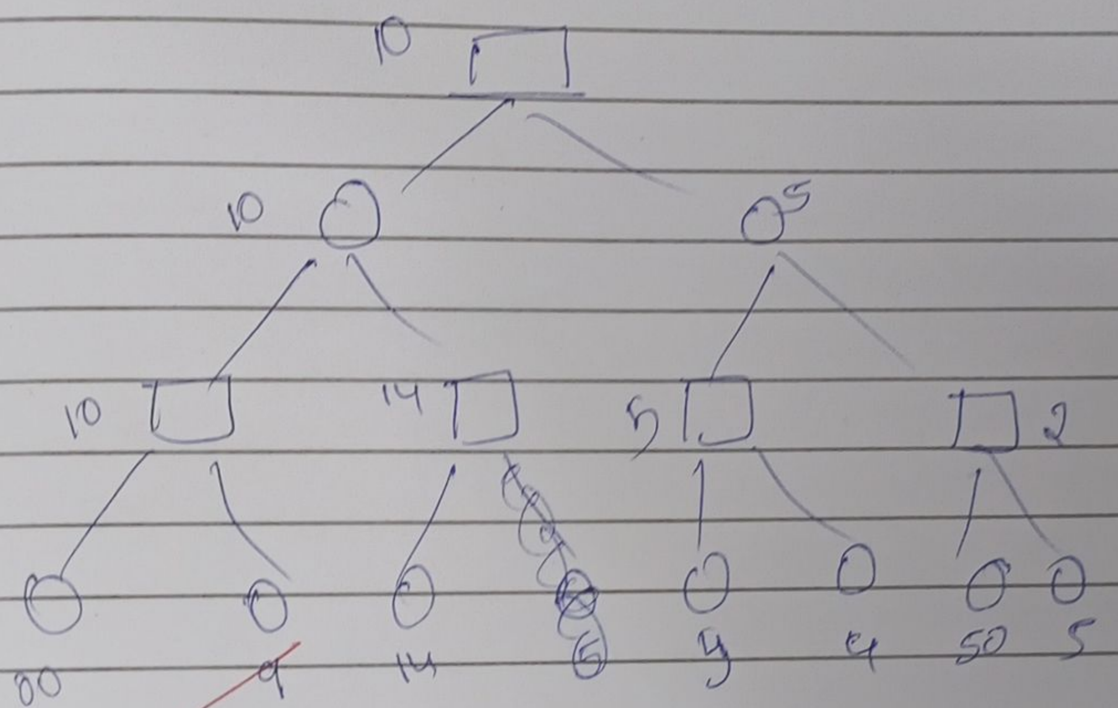
Alpha-Beta Pruning

```
Function minimax (node, depth, isMaximisingPlays
                  alpha, beta):

if node is a leaf node
    return value of the node

if isMaximisingPlayer:
bestVal = -INFINITY

for each child node:
    value = minimax(node, depth+1, false,
                    alpha, beta)
    bestVal = max(bestVal, value)
    alpha = max(alpha, bestVal)
    if beta <= alpha:
        break
return bestVal
else:
    bestVal = INFINITY
    for each child node:
        value = minimax(node, depth+1, true,
                        alpha, beta)
        bestVal = min(bestVal, value)

    beta = min(beta, bestVal)
    if beta <= alpha
        break
    return bestVal

minimax(0, 0, true, -INFinity, +INFINITY)
```

Date ___/___/___
Page ________

example





17/12/24