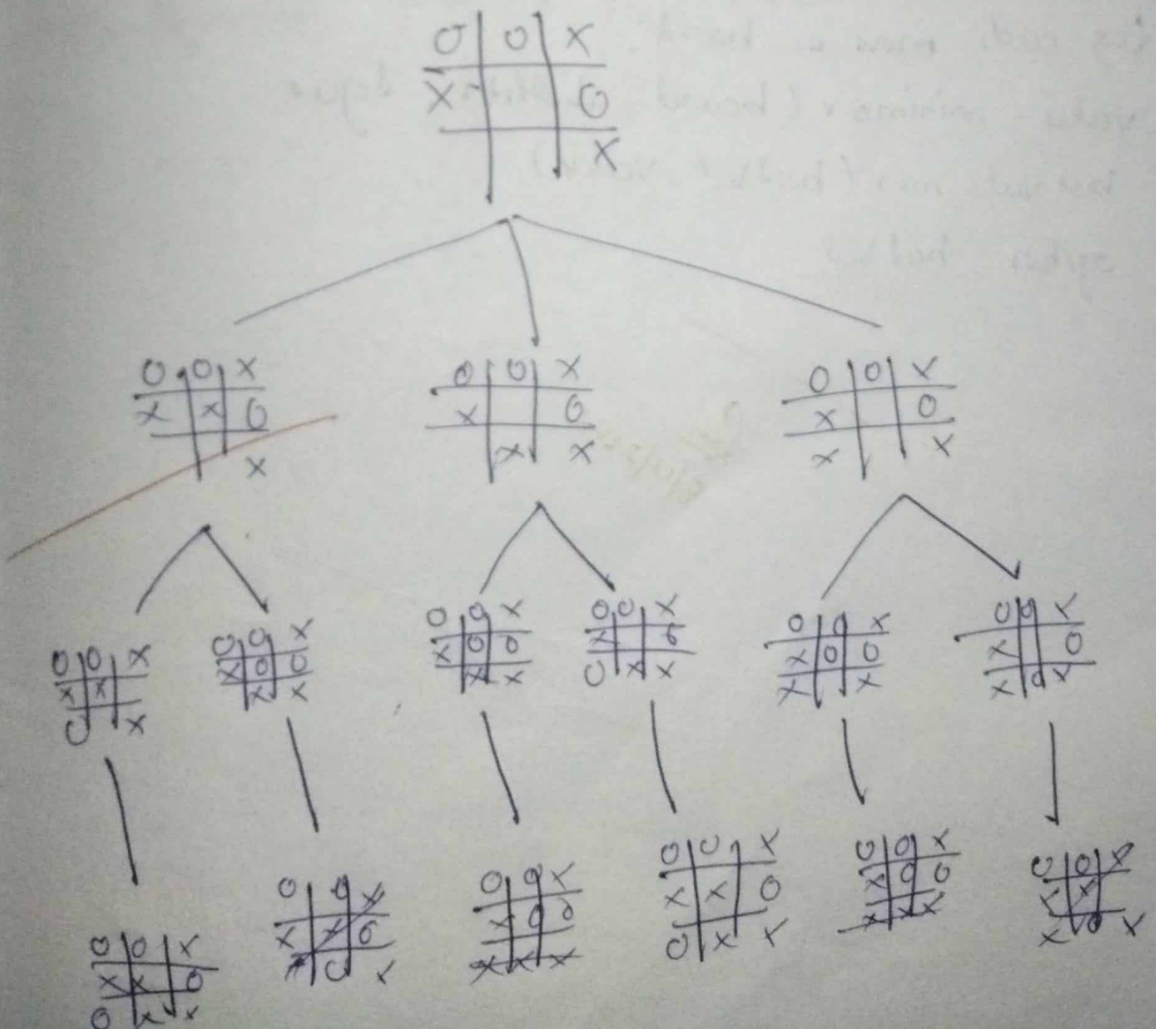
Tic Tac Toe

-> Artificial Intelligence (AI) can be used for this game in order to play only single player. That is human against Computer.

-> Minimax Algorithm is used implemented

-> It

-> To find / calculate the all possible moves available for the computer



Algorithm

function minimax (board, depth, isMaximizingPlayer):

if current board state is a terminal state:
    return value of the board

if isMaximizing Player:

    bestVal = -INFINITY
    for each move in board:

        value = minimax (board, depth+1, false)
        bestVal = max (bestVal, value)
    return bestVal

else:
    bestVal = +INFINITY
    for each move in board:
        value = minimax (board, depth+1, true)
        bestVal = min (bestVal, value)
        return bestVal

3/11/2023

15/10/2023

AI in gaming refers to the integration of artificial intelligence.

Type of AI gaming

1) Rule based AI
2) Finite State Machine
3) Path finding AI
4) Machine learning AI
5) Behaving Trees
6) Reinforcement learning

15/9/23

Puzzle

```
def bfs(src, target)
    queue = []
    queue.append(src)
    exp = []
    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)
        print(source)
        if source == target:
            print("Success")
            return
        poss_moves_to_do = []
        poss_moves_to_do = possible_moves(source, exp)
        for move in poss_moves_to_do:
            if move not in exp and move not in queue:
                queue.append(move)

def possible_moves(state, visited_states):
    b = state.index(0)
    d = []
```

LAB program 6/12/2023

1) Analyse Iterative Deepening Search algorithm. Demonstrate how 8 Puzzle problem could be solved using this algorithm implement the same

* Iterative Deepening Search (IDS) is a combination of DFS & BFS
* IDS performs a series of depth limited searches gradually increasing the depth limit in each iteration until the goal state is reached

src = [1, 2, 3, -1, 4, 5, 6, 7, 8]

target = [1, 2, 3, 4, 5, -1, 6, 7, 8]

source :

$$\begin{bmatrix} 1 & 2 & 3 \\ -1 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

goal :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & -1 \\ 6 & 7 & 8 \end{bmatrix}$$

Solution

$$\begin{bmatrix} -1 & 2 & 3 \\ 1 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & - & 5 \\ 6 & 7 & 8 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 6 & 4 & 5 \\ - & 7 & 8 \end{bmatrix}$$

Solution

Row $\begin{bmatrix} 1 & 2 & 3 \\ - & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$

Broad $\begin{bmatrix} 1 & 2 & 3 \\ 6 & 4 & 5 \\ - & 7 & 8 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & - & 5 \\ 6 & 7 & 8 \end{bmatrix}$ $\begin{pmatrix} - & 1 & 3 \\ 4 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 & 3 \\ 6 & 4 & 5 \\ - & 7 & 8 \end{pmatrix}$

10 to 12

Sir/m

```
class Eightpuzzle:
def __init__(self, initial_state, goal_state):
 self.initial_state = initial_state
  self.goal_state = goal_state
  self.actions = [(-1,0), (1,0), (0,-1), (0,1)]

  def is_goal_state(self, state):
      return state == self.goal_state

  def get_successor(self, state):
        successor = []
        empty_tile_index = state.index(0)

    for action in self.actions:
        new_index = empty_tile_index + action[0] +
                                       action[1]

    if 0 <= new_index < 9:
          new_state = list(state)
          new_state[empty_tile_index],
        new_state[new_index] = new_state[new_index],
        new_state[empty_tile_index]
          successor.append(new_state)

        return successor
```

```
def depth_limited_search(self, state, limit):
    if self.is_goal_state(state):
        return [state]

    if(limit==0)
        return none
    for successor in self.get_successors(state):
        result = self.depth_limited_search(successor, limit-1)
        if result is not none:
            return [state] + result

    return none:
```

A* algorithm

```
def print_b(src):
    stak = src.copy()
    stak[stak.index(-1)]=''
    print(f"  {stak[0]} {stak[1]} {stak[2]}
          {stak[3]} {stak[4]} {stak[5]}
          {stak[6]} {stak[7]} {stak[8]}
                  ...
          ")

def h(stak, target):          # for heuristic value
    cant=0
    i=0
    for j in stak:
        if stak[i] != target[i]:
            cant = cant+1
        i=i+1
    return cant

def astar(stak, target): # Add inputs
    stak = [src]
    g=0
    visited_staks=[]
    while len(staks):
        print("Level:",g,"")
        move = []
        for stak in staks:
            visited.append(stak)

    print_b(stak)
    if stak == target:
        print("Success")
        return
    moves += [move for move in possible_moves(
        stak, visited, staks) if move not in moves]
    costs = [g + h(move, target) for move in moves]
```

```
stak = [moves[i]
for i in range(len(moves)): if costs[i] == min(costs)]

gt = 1
print("Fail")

def possibl_moves(stak, visited_stak):

b = stak.index(-1)

d = []
if b-3 in range(a):
    d.append('u')
if b not in [0, 3, 6];
    d.append('l')
if not in [2, 5, 8];
    d.append('r')
if b+3 in range

pos_moves = []
for m in d
    pos_moves.append(gen(stak_m,b))
return [move for move in pos_moves if move not
                                    in visited_stak]

def gn(stak_m,b):
    temp = stak.copy()

if m == 'u'
    temp[b-3], temp[b] = temp[b], temp[b-3]

if m == 'l'
    temp[b-1], temp[b] = temp[b], temp[b+1]

if m == 'r'
    temp[b+1], temp[b] = temp[b], temp[b+1]
```

if m == 'd':

tmp[b+3], tmp[b] = tmp[b], tmp[b+3]

return tmp

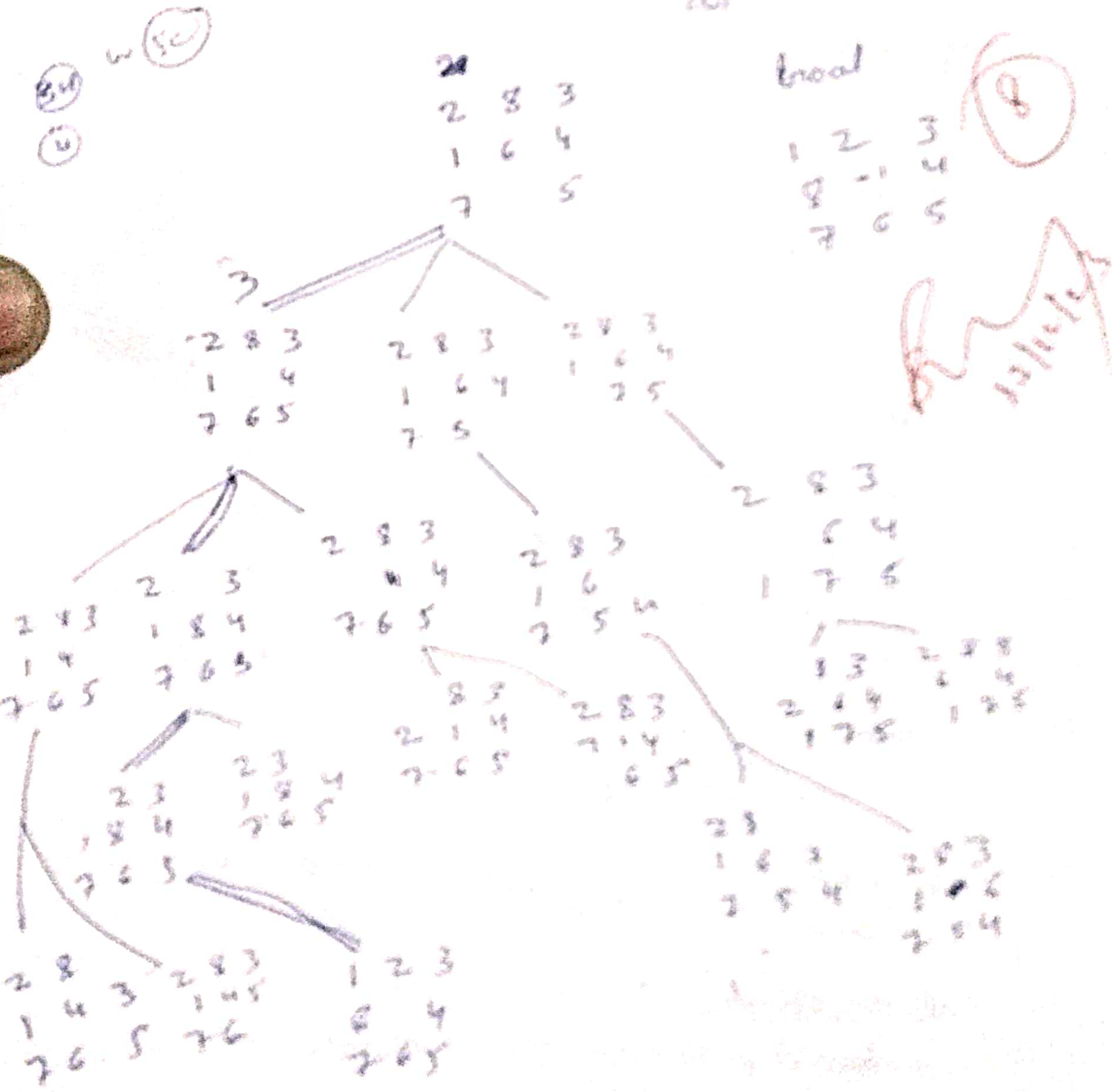
2	8	3
1	6	4
7	-1	5

src = [2, 8, 3, 1, 6, 4, 7, -1, 5]

target = [1, 2, 3, 8, -1, 4, 7, 6, 5]

astar(src, target)

$f = g + h$



now first it will compare
goal state and current state
It count no of misplaced element
and it add heuristic value and
cost path value

Propositional logic

$KB = (A \vee C) \wedge (B \vee \neg C)$

$\alpha = (A \vee B)$

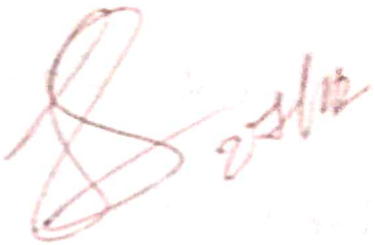
A	B	C	A ∨ C	B ∨ ¬C	KB	α
T	T	T	T	T	T	T
T	T	F	T	T	T	T
T	F	T	T	F	F	T
T	F	F	T	T	T	T
F	T	T	T	T	T	T
F	T	F	F	T	F	T
F	F	T	T	F	F	F
F	F	F	F	T	F	F

```
def evaluate_expression(P, Q, R):
    expression_result = (~P & Q) | (~R | P)
                      = (P and Q) and (not R or P)
    return expression_result

def generate_truth_table():
    print("P | Q | R | Expression(P&Q) | Query (P^R)")
    print("-----------------------------------")

    for P in [True, False]:
        for Q in [True, False]:
            for R in [True, False]:
                expression_result = evaluate_expression(P, Q, R)
                query_result = P and R
                if expression_result and not query_result:
                    return False
    return True
```

```
def main():
    generate_truth_table()
    if query_entails_knowledge():
        print("\n Query entails the knowledge")
    else
        print("\n Query does not entail the knowledge")
if __name__ == "__main__"
    main()
```



Program 7:

Create a knowledgebase using propositional logic and prove the given query using resolution

Express

R ∨ P R ∨ Q ¬R ∨ P ¬R ∨ Q

(R ∨ not P) (R ∨ not Q) (not R ∨ P)

(not R ∨ Q)

Step	Clause	Derivation
	R ∨ P	Given
	R ∨ Q	Given
	¬R ∨ P	Given
	¬R ∨ Q	Given
	¬R	Negated conclusion.

Resolved R ∨ P and ¬R ∨ P which is in the form null. A contradiction is found when ¬R is assumed as true. Hence, R is true

```
import re

def main(rules, goal):
    rules = rules.split('\n')
    steps = resolve(rules, goal)
    print('\nStep\t| Clauses\t| Derivation\t')
    print('-' * 30)
    i = 1
    for step in steps:
        print(f"{i}.\t| {step}\t| {steps[step]}")
        i += 1

def negate(term):
    return f'~{term}' if term[0] != '~' else term[1]

def resolve(clauses):
    if len(clauses) > 2:
        t = split_terms(clauses)
        return f'{t[1]} {t[0]}'
    return ''

def split_terms(rule):
    exp = '(~*[PQRS])'
    terms = re.findall(exp, rule)
    return terms

def contradiction
```

Implement unification in FOL

```
def unify(expr1, expr2):
    func1, args1 = expr1.split('(', 1)
    func2, args2 = expr2.split('(', 1)

    if func1 != func2:
        print("Not unified")
        return None
    args1 = args1.rstrip(')').split(',')
    args2 = args2.rstrip(')').split(',')

    substitution = {}

    for a1, a2 in zip(args1, args2):
        if a1.islower() and a2.islower() and a1 != a2:
            substitution[a1] = a2
        elif a1.islower() and not a2.islower():
            substitution[a1] = a2
        elif not a1.islower() and a2.islower():
            substitution[a2] = a1
        elif a1 != a2:
            print("Expression cannot be unified.")
            return None
    return substitution
```

```
def apply_substitution(expr, substitution):
    for key, value in substitution.items():
        expr = expr.replace(key, value)
    return expr

if __name__ == "__main__":
    expr1 = input("Enter the first expression: ")
    expr2 = input("Enter the second expression: ")

    substitution = unify(expr1, expr2)

    if substitution:
        print("The substitution are:")
        for key, value in substitution.items():
            print(f"{key} / {value}")

        expr1_result = apply_substitution(expr1, substitution)
        expr2_result = apply_substitution(expr2, substitution)

        print(f"Unified expression 1: {expr1_result}")
        print(f"Unified expression 2: {expr2_result}")
```

10/1

OP

First expression: knows(John, x)
Second expression: knows(y, mother(y))
-> substitution are
* John/y
* x/mother(y)

```
any
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token A B NL
%%
stmt : A A A A A B NL { printf("valid string\n");
        exit(0); }
     ;
S : S A
  | ;
%%
int yyerror(char *msg)
{ printf("invalid string\n");
  exit(0);
}
main()
{ printf("enter the string\n");
  yyparse();
}
```

```
%{
#include "y.tab.h"
%}
%%
[aB]+ {yylval = A; }
[bB] {yylval = B; }
\n { return NL; }
. { return yytext[0]; }
%%
int yywrap()
{
return 1;
}
```

Output

Enter the string

aaabb

Invalid expression

Write a Yacc program to generate syntax tree for a given arithmetic expression

P1.l

```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext); return digit;}
[\t]
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

p1.y

```
%{
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct tree_node
{ char val[10];
  int lc;
  int rc;
};
```

```
int ind;
typedef tree_node syn_tree [100];
void my_print_tree (int cur_ind);
int mknode (int lc, int rc, char val [10]);

%}
%token digit

%%
S:E { my_print_tree ($1); }
;
E:E'+'T { $$ = mknode ($1, $3, "+"); }
|T { $$ = $1; }
;
T:T'*'F { $$ = mknode ($1, $3, "*"); }
|F { $$ = $1; }
;
```

Output

```
root@DESKTOP-HUVACGE:~# ./a.out
Enter an expression
2+3*5
Operator Node -> Index: 4, Value: +, Left child Index:
0 Right child index: 3
Root node -> Index 4, value: +
```

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token A B NL
%%
stmt: S NL { printf("valid string\n");
exit(0); }
;
S: A S B
|;
%%
int yyerror(char *msg)
{ printf("invalid string\n");
  exit(0);
}
main()
{ printf("enter the string\n");
  yyparse();
}

%{
#include "y.tab.h"
%}
%%
```

[OR]

Python

```
[bB] { return B; }
\n { return NL; }
  { return yytext[0]; }

%%
int yywrap()

{ return 1;
}
```

11/1/24

Program 7

FOL to CNF

```
import re
def fol_to_cnf(fol):
    statement = fol.replace("<=>", "_")
    while '_' in statement:
        i = statement.index('_')
        new_statement = '[' + statement[:i] + '=>' +
                        statement[i+1:] + ']&[' +
                        statement[i+1:] + '=>' +
                        statement[:i] + ']'
    statement = new_statement
    statement = statement.replace("=>", "-")
    expr = '\[([^\]]+)\]'
    statements = re.findall(expr, statement)
    for i, s in enumerate(statements):
        if '[' in s and ']' not in s:
            statements[i] += ']'
    for s in statements:
        statement = statement.replace(s, fol_to_cnf(s))
```

Exercises

① animal (y) <=> loves (x, y)

[animal(y) | loves (x, y)] & [¬loves (x, y) | animal(y)]

② ∀x [∀y [animal (y) => loves (x, y)] => [∃z [loves (z, x)]]]

=>

[animal (G (x)) & ¬ loves (x, G(x))] |
[loves (F(x), x)]

(If y is not an animal. then x loves y

Steps

① Elimination biconditionals and implications

② Move ¬ inwards

③ Standardize variables

④ Drop universal quantifiers

⑤ Distribute.

=> (P ∧ Q)

¬ P ∨ Q)

<=>

P => Q

Q => P

Create a knowledgebase consisting of FOL logic statement and prove the given query using forward chaining.

①

kb_ = KB()
kb_.tell('king(x) & greedy(x) => evil(x)')
kb_.tell('king(John)')
kb_.tell('greedy(John)')   T T -> T
kb_.tell('king(Richard)')
kb_.tell('evil(x)')

Querying evil(x):
1. evil(John).

②
tell(missile(x) => weapon(x))
tell(missile(M1))
tell(enemy(x, America) => hostile(x))
tell(american(west))
tell(enemy(Nono, America))
tell(owns(Nono, M1))
tell.(missile(x) & owns(Nono, x) => sells(West, x, Nono))
tell.(american(x) & weapon(y) & sells(x, y, z) & hostile(z) => criminal(x))

Entryque

Criminal(x)

Querying criminal(x):
1. criminal(West)

FOL

All facts :

① missile(M_1)

② sells(West, M_1, Nono)

③ hostile(Nono)

④ owns(Nono, M_1)

⑤ weapon(M_1)

⑥ criminal(west)

⑦ american(West)

⑧ enemy(Nono, America)