this function specifies the winner when pattern is achieved.

6. We have another function to print board.

Program

```
board = [' ' for x in range(10)]

def insert letter(letter, pos):
    board[pos] = letter

def space Is Free(pos):
    return board[pos] == ' '

def printBoard(board):
    print('   |   |')
    print(' ' + board[1] + ' | ' + board[2] + ' | ' board[3])
    print('   |   |')
    print('-----------')
    print(' ' + board[4] + ' | ' + board[8] + ' | ' +
                                        board[9])
    print('   |   |')
```

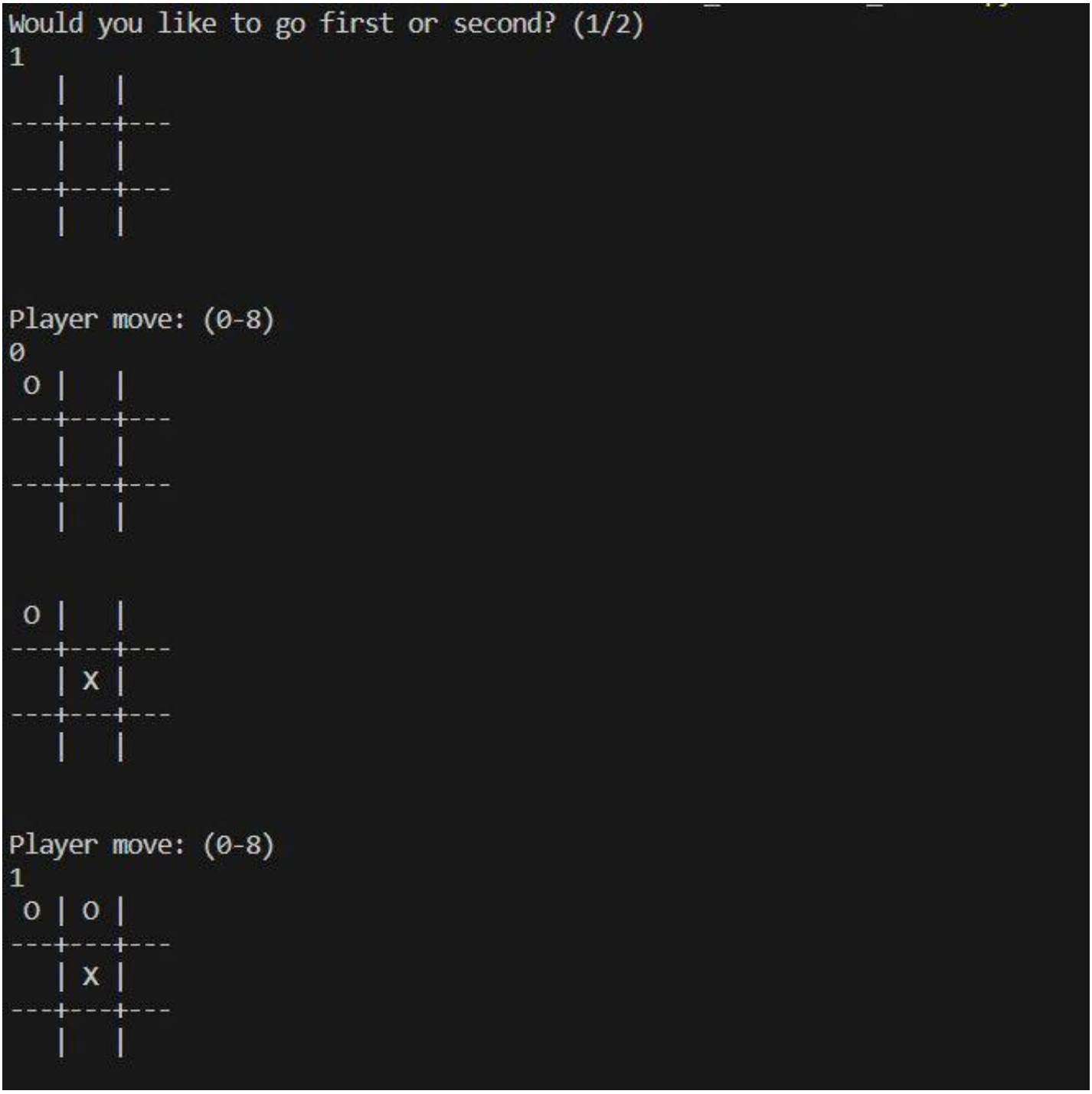
```
def iswinner(bo, le):
    return (bo[7] == le and bo[8] == le and
            bo[9] == le or (bo[4] == le and
            bo[5] == le and bo[6] == le) or
            (bo[1] == le and bo[2] == le and bo[3] == le) or
            (bo[1] == le and bo[4] == le and bo[7] == le) or
            (bo[2] == le and bo[5] == le and bo[8] == le) or
            (bo[3] == le and bo[6] == le and bo[9] == le) or
            (bo[1] == le and bo[5] == le and bo[9] == le) or
            (bo[3] == le and bo[5] == le and bo[7] == le))
```

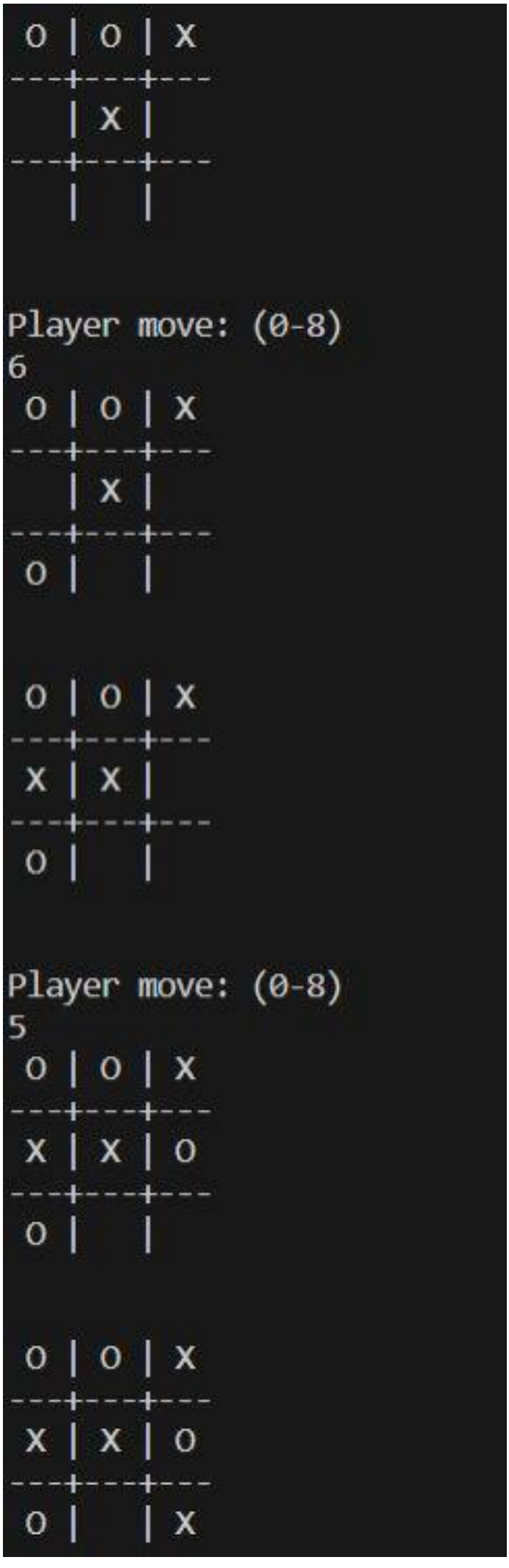
```
def playgame():
    run = True
    while run:
        move = input("Select a position to play")
        move = int(move)
        if move > 0 and move < 10:
            if spaceIsFree(move):
                run = False
                insertLetter('X', move)
            else:
                print('Space is occupied')
        else:
            print('Please a no. within range')
```

```
def comparemove():
    possiblemoves = [x for x, letter in enumerate(board)
                     if letter == ' ' and x != 0]

    move = 0

    cornersopen = []
    for i in possiblemoves:
        if i in [1,3,7,9]:
            cornersopen.append(i)
```


Would you like to go first or second? (1/2)
1
Player move: (0-8)
0
Player move: (0-8)
1


O | O | X
---+---+---
| X |
---+---+---
| |
Player move: (0-8)
6
O | O | X
---+---+---
| X |
---+---+---
O | |
O | O | X
---+---+---
X | X |
---+---+---
O | |
Player move: (0-8)
5
O | O | X
---+---+---
X | X | O
---+---+---
O | |
O | O | X
---+---+---
X | X | O
---+---+---
O | | X

code:

```
import numpy as np
import pandas as pd
import os

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 = []
    if b not in [0, 1, 2]:
        d.append('u')
```

```
if b not in [6,7,8]
    d.append('d')
if b not in [0,3,6]:
    d.append('l')
if b not in [2,5,8]:
    d.append['r']

pon_moves_it_can = []

for i in d:
    pon_moves_it_can.append(gen(state, i, b))

return [moves_it_can for moves_it_can in
        pon_moves_it_can if move_it_can not in
            visited_state]

def gen(state, m, b):
    temp = state.copy()
    if m == 'd':
        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]
    return temp.
```

src = [1, 2, 3, 4, 5, 6, 0, 7, 8]

target = [1, 2, 3, 6, 4, 6, 7, 8, 0]

bfs (src, target).

Output:

```
1 | 2 | 3
4 | 5 | 6
0 | 7 | 8

1 | 2 | 3
0 | 5 | 6
4 | 7 | 8

1 | 2 | 3
4 | 5 | 6
7 | 0 | 8

0 | 2 | 3
1 | 5 | 6
4 | 7 | 8

1 | 2 | 3
5 | 0 | 6
4 | 7 | 8

1 | 2 | 3
4 | 0 | 6
7 | 5 | 8

1 | 2 | 3
4 | 5 | 6
7 | 8 | 0

success
```

8 puzzle problem using BFS

```
import numpy as np
import pandas as pd
import os
def bfs(src, target):
    queue = []
    queue.append(src)
    exp = []
    while len(queue) > 0:
        source = queue.pop(0)
        exp.append(source)
        if source == target:
            print("Success")
            return
    poss_moves_to_do = []
    poss_moves_to_do = possible_moves(source, exp)
    for now in poss_moves_to_do:
        if now not in exp and now not in
        queue:
            queue.append(now)
def possible_moves(state, visited_states):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
        d.append('u')
```

```
if b not in [6,7,8]:
    d.append('d')
if b not in [0,3,6]:
    d.append('l')
if b not in [2,5,8]:
    d.append('r')

pos_move_it_con = []
for i in d:
pos_move_it_con.append(gen(state,i,b))

return [move_it_con for move_it_con in
    pos_move_it_con if move_it_con not in
    visited_state]

def gen(state,m,b):
    tmp = state.copy()
    if m == 'd':
temp[b+3], tmp[b] = tmp[b], tmp[b+3]
    if m == 'u':
temp[b-3], tmp[b] = tmp[b], temp[b-3]
    if m == 'l':
tmp[b-1], tmp[b] = tmp[b], tmp[b-1]
    if m == 'r':
tmp[b+1], tmp[b] = tmp[b], tmp[b+1]
    return tmp
```

src = [1, 2, 3, 4, 5, 6, 0, 7, 8]

target = [1, 2, 3, 4, 5, 6, 7, 8, 0]

bfs (src, target)

output:

```
Success!! It is possible to solve 8 Puzzle problem
Path: [[1, 2, 3, 0, 4, 6, 7, 5, 8], [1, 2, 3, 4, 0, 6, 7, 5, 8], [1, 2, 3, 4, 5, 6, 7, 0, 8], [1, 2, 3, 4, 5, 6, 7, 8, 0]]
```

08/12/23

8 puzzle problem

```
def id_dfs(puzzle, goal, get_moves)
    import itertools

def dfs(route, depth):
    if depth == 0:
        return
    if route[-1] == goal:
        return route

    for move in get_moves(route[-1]):
        if move not in route:
            next_route = dfs(route + [move], depth-1)

            if next_route:
                return next_route

    for depth in itertools.count():
        route = dfs([puzzle], depth)

        if route:
            return route

def possible_move(state):
    b = state.index(0)
    d = []
    if b not in [0,1,2]:
        d.append('u')
```

```
if b not in [6,7,8]:
    d.append('d')

if b not in [0,3,6]:
    d.append('l')

if b not in [2,5,8]:
    d.append('r')
pos_moves = []
    for i in d
pos_moves.append(generate(state, i, b))
    return pos_moves.

def generate(state, m, b):
    temp = state.copy()
    if m == 'd':
temp[b+3], temp[b] = temp[b], temp[b+3]
    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]
    return temp
initial = [1,2,0,4,5,
```

```
Enter the start state matrix

1 2 3
4 5 6
_ 7 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _




 \'/

1 2 3
4 5 6
_ 7 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/

1 2 3
4 5 6
7 8 _
```

22/12/23

Vacuum cleaner:

```
def clean_room(floor, room_row, room_col):
    if floor[room_row][room_col] == 1:
        print("Cleaning room at ({room_row+1}, {room_col+1}) (room was dirty)")
        floor[room_row][room_col] = 0
        print("Room is clean now")
    else:
        print("Room at ({room_row+1}, {room_col+1}) is already clean")

def main():
    rows = 2
    cols = 2
    floor = [[0,0], [0,0]]
    for i in range(rows):
        for j in range(cols):
            status = int(input("Enter clean status for room at ({i+1}, {j+1}) (1 for dirty, 0 for clean): "))
            floor[i][j] = status
```

```
    for i in range (num):
        for j in range (col):
            clean_room (floor, i, j)
    print ("Returning to room at (1,1) to check
            if it has become dirty again:")
    clean_room (floor, 0, 0)
if __name__ == "__main__":
    main()
```

Output:

```
Enter initial state for 4 rooms
Room 0 state : 0
Room 1 state : 1
Room 2 state : 1
Room 3 state : 0

Room 0 is already clean.
vacuum Room 1 ...
cleaning Room 2 ...
Room 3 is already clean.
Room 0 is already clean.

All rooms have been processed.
```

25/1/24

```
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 1
[('Room 1', 1), ('Room 2', 1)]
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Cleaning Room 2 (Room was dirty)
Room 2 is now clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.
```

24/11/23

Knowledge base entailment

Entailment refers to the logical relationship b/w a KB (a set of statements or rules) & a query.

If KB entails a statement, it means that if the statements in the KB are true, then query must also be true.

$KB \models Q$.

Knowledge base resolution.

The resolution rule involves taking two clauses that contain complementary literals and resolving them to produce a new clause.

Knowledge base entailment

- If it's raining (p) then ground is wet.
- If the ground is wet (Q), then the plants will grow (R).
- It's not the case that plants will grow (-R)

$P \rightarrow Q$
$Q \rightarrow R$.
Not r.

Eg: p - It's sunny.
q - I go for picnic
r - I carry an umbrella.

$P \rightarrow R$

code:

```
from sympy import symbols, And, Not, Implies,
    satisfiable

def create_knowledge_base():
    p = symbols('p')
    q = symbols('q')
    r = symbols('r')

    knowledge_base = And(
        Implies(p, q),
        Implies(q, r),
        Not(r)
        )

    return knowledge_base

def query_entails(knowledge_base, query):
    entailment = satisfiable(And(knowledge_base, Not(query)))

    return not entailment

if __name__ == "__main__":
    kb = create_knowledge_base()
    query = symbols('p')
    result = query_entails(kb, query)
    print("Knowledge Base: kb")
    print("Query:", query)
    print("Query entails Knowledge Base:", result)
```

Smita
29/12/23

```
Knowledge Base: ~r & (Implies(p, q)) & (Implies(q, r))
Query: p
Query entails Knowledge Base: False
```

Output:

Knowledge Base: ~r & (Implies (kb) & Implies(...

Query: p

Query entails knowledge base: False.

* Knowledge base resolution:

Code:

```
def negate_literal(literal):
    if literal[0] == '~':
        return literal[1:]
    else:
        return '~' + literal

def resolve(c1, c2):
    resolved_clause = set(c1) | set(c2)
    for literal in c1:
        if negate_literal(literal) in c2:
            resolved_clause.remove(literal)
            resolved_clause.remove(negate_literal(literal))
    return tuple(resolved_clause)

def resolution(knowledge_base):
    while True:
        new_clauses = set()
```

```
for i, c1 in enumerate (knowledge_base):
    for j, c2 in enumerate (knowledge_base):
        if i != j:
            new_clause = resolve (c1, c2)
            if len (new_clause) > 0 and
                new clause not in knowledge base
                new_clauses.add (new_clause).

        if not new_clauses:
            break.

knowledge_base1 = new_clauses
return knowledge_base

if __name__ == "__main__":
    kb = {('p', 'q'), ('~p', 'r'), ('~q', '~r')}
    result = resolution (kb)
    Print ("original knowledge Base : ", kb)
    Print ("Solved knowledge Base : ", result):
```

Step	Clause	Derivation
1.	Rv~P	Given.
2.	Rv~Q	Given.
3.	~RvP	Given.
4.	~RvQ	Given.
5.	~R	Negated conclusion.
6.		Resolved Rv~P and ~RvP to Rv~R, which is in turn null.

A contradiction is found when ~R is assumed as true. Hence, R is true.

19/01/2024

unification

code

```
import re
def getAttributes(expression):
    expression = expression.split(",")[1:]
    expression = ",".join(expression)
    expression = expression[:-1]
    ____ = re.split("(?<!\(.),(?!.\))", expression)
    return expression

def unify(exp1, exp2):
    if exp1 == exp2:
        return []
    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            return False
    if isConstant(exp1):
        return [(exp1, exp2)]
    if isConstant(exp2):
        return [(exp2, exp1)]
    if isVariable(exp1):
        if checkOccurs(exp1, exp2):
            return False
        else
            return [(exp2, exp1)]
```

```
if getInitialPredicate(exp1) != getInitialPredicate(exp2)
    print("Predicate don't match!");
    return False

attributeCount1 = len(getAttributes(exp1))
_____2 = _____2.

if attributeCount1 != attributeCount2:
    return False

head1 = getFirstPart(exp1)
head2 = _____(exp2)

initialSubstitution = unify(head1, head2)

if not initialSubstitution:
    return False

if attributeCount1 == 1:
    return initialSubstitution

tail1 = getRemainingPart(exp1)
tail2 = _____(exp2)

if initialSubstitution != []
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution).

remainingSubstitution = unify(tail1, tail2)

if not remainingSubstitution
    return False

initialSubstitution.extend(remainingSubstitution)
return initialSubstitution

exp1 = "knows(x)"
exp2 = "knows(Richard)"
substitutions = unify(exp1, exp2)
print("substitutions")
print(substitutions)
```

Output:

[('x', 'Richard')]

inp1 = 'knows (A, x)'

inp2 = 'knows (y, mother (y))'

Output: Substitutions:

[('A', 'y'), ('mother (y)', 'x')].

```
Substitutions:
[('A', 'y'), ('Y', 'x')]
```

19/01/2024

Conversion of FOL to CNF

```
from sympy import symbols, AND, OR, Implies,
    Not, to_cnf

def eliminate_implications(formula):
    return formula.subs(Implies(P,Q), OR(Not(P),Q))

def move_negations_inwards(formula):
    return formula.simplify()

def fol_to_cnf(fol_formula):
    formula_step1 = eliminate_implications(fol_formula)

    formula_step2 = move_negations_inwards
```

Output:

Enter a first order logic: P2(-a/a)

FOL formula: AND(P or Not(a), T))

CNF formula: or (And(P,T) and (P, ~a))

```
~bird(x)|~fly(x)
[~bird(A)|~fly(A)]
```

Create a KB consisting of FOL and prove given query using forward reasoning.

```
class KnowledgeBase:
    def __init__(self):
        self.statement = []

    def add_statement(self, statement):
        self.statements.append(statement)

    def fol_forward_reasoning(self, query):
        working_memory = set()
        unchanged = False

        while not unchanged:
            unchanged = True

        return query in working_memory
```

Example

```
kb = KnowledgeBase()
kb.add_statement({"P", "Q"}), kb.add_statement({"Q", "R"})

query = {"Q"}
result = kb.fol_forward_reasoning(query)
if result:
    print("Query is True")
```

else
    print ("Query is false")

output:

Query is false.

25/1/24

```
Querying criminal(x):
        1. criminal(West)
All facts:
        1. missile(M1)
        2. weapon(M1)
        3. enemy(Nono,America)
        4. owns(Nono,M1)
        5. hostile(Nono)
        6. criminal(West)
        7. american(West)
        8. sells(West,M1,Nono)
```