Artificial Intelligence (AI)
Lab Sheet No: 2

IOE, Thapathali Campus

December, 2025

Introduction

Constraint programming is a useful tool in formulating and solving problems that can be defined in terms of constraint among a set of variables. In fact real world problems are defined in terms of some variables that bear some constraints. Finding a set of variables that are within the constraints given (or observed) is a solution to this problem.

Let us consider a problem that can be represented by some relations of the variables x, y and z. We have a domain D_x, D_y, D_z from which the variables can take a value. The constraint is given by a set C and may have a number of constraints C_1, C_2, C_3 etc each relating some or all of the variables x, y, z. Now a solution (or solutions) to the problem is a set of the problem is a set $d_x \in D_x$, $d_y \in D_y$, $d_z \in D_z$, and all the constraints of set C are satisfied.

Crypto Arithmetic Problem

The Crypto Arithmetic Problem is yet another constraint satisfaction problem. We have to assign numeric values (0 through 9) to the alphabets of the given words in such a way that the sum of the two words equals the third.

For example: SEND + MORE = MONEY

We have to assign values to the individual alphabets in such a way the arithmetic rules are followed, a trivial solution will be assign zeros to all but we have a constraint, no two alphabets should be assigned with the same number.

Now, domain for alphabet is given by:

$$S, E, N, D, M, O, R, Y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The constraints are:

$$\begin{aligned} D + E &= Y + 10C_1 \\ N + R + C_1 &= E + 10C_2 \\ E + O + C_2 &= N + 10C_3 \\ S + M + C_3 &= O + 10C_4 \\ M &= C_4 \\ C_1, C_2, C_3, C_4 &\in \{0, 1\} \end{aligned}$$

We also have the constraint that no two alphabets should be assigned to the same number.

PROGRAM: 1

```
from itertools import permutations

def solution():
    """
    Solve the cryptarithmetic puzzle: SEND + MORE = MONEY
    Each letter represents a unique digit (0-9).
    """
    # Try all permutations of 8 unique digits for S, E, N, D, M, O,
       R, Y
    for perm in permutations(range(10), 8):
        S, E, N, D, M, O, R, Y = perm

        # S and M cannot be 0 (leading digits in SEND and MORE)
        if S == 0 or M == 0:
            continue

        # C4 must be 1 (carry from the thousands column)
        C4 = 1

        # Solve the addition constraints
        # D + E = Y + 10*C1
        C1 = (D + E) // 10
        if (D + E) % 10 != Y:
            continue

        # N + R + C1 = E + 10*C2
        C2 = (N + R + C1) // 10
        if (N + R + C1) % 10 != E:
            continue

        # E + O + C2 = N + 10*C3
        C3 = (E + O + C2) // 10
        if (E + O + C2) % 10 != N:
            continue

        # S + M + C3 = O + 10*C4
        if S + M + C3 != O + 10 * C4:
            continue

        # M = C4
        if M != C4:
            continue

        # All constraints satisfied
        return [S, E, N, D, M, O, R, Y]

    return None

# Find and display the solution
```

```
result = solution()
if result:
    S, E, N, D, M, O, R, Y = result
    print(f"Solution found:")
    print(f"S={S}, E={E}, N={N}, D={D}, M={M}, O={O}, R={R}, Y={Y}")
    print()

    # Display the addition
    SEND = 1000*S + 100*E + 10*N + D
    MORE = 1000*M + 100*O + 10*R + E
    MONEY = 10000*M + 1000*O + 100*N + 10*E + Y

    print(f"  {SEND}")
    print(f"+ {MORE}")
    print(f"-------")
    print(f" {MONEY}")
else:
    print("No solution found.")
```

Assignment (1)

Make yourself a crypto arithmetic problem as above and find the solution.

Eight Queens Problem

The Eight queens problem is a constraint satisfaction problem. The task is to place eight queens in the 64 available squares in such a way that no queens attack each other. So, the problem can be formulated with variables $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$ and $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$; xs represents the rows and ys the column. Now, a solution to this problem is to assign values for x and y such that the constraint is satisfied.

The problem can be formulated as $P = \{(x_1, y_1), (x_2, y_2), \ldots, (x_8, y_8)\}$ where (x_1, y_1) gives the position of the first queen, etc.

So, it can be clearly seen that the domains of x_i and y_i are:

$$D_x = \{1, 2, 3, 4, 5, 6, 7, 8\} \quad \text{and} \quad D_y = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

respectively.

The constraints are:

1. No two queens should be in the same row. That is, $y_i \neq y_j$ for $i = 1$ to 8; $j = 1$ to 8; $i \neq j$
2. No two queens should be in the same columns. That is, $x_i \neq x_j$ for $i = 1$ to 8; $j = 1$ to 8; $i \neq j$
3. There should not be two queens placed on the same diagonal line. That is, $(y_i - y_j) \neq \pm(x_i - x_j)$

Now, a solution to this problem is an instance of P wherein the above mentioned constraints are satisfied.

PROGRAM: 2

```
def is_safe(board, row, col):
    """Check if placing a queen at (row, col) is safe"""
    # Check column
    for i in range(row):
        if board[i] == col:
            return False

    # Check upper-left diagonal
    for i in range(row):
        if abs(board[i] - col) == abs(i - row):
            return False

    return True

def solve_queens(board, row, n=8):
    """Recursively solve the n-queens problem"""
    if row == n:
        return True

    for col in range(1, n + 1):
        if is_safe(board, row, col):
            board[row] = col
            if solve_queens(board, row + 1, n):
                return True
            board[row] = 0

    return False

def find_all_solutions(row, n=8, board=None, solutions=None):
    """Find all possible solutions to the n-queens problem"""
    if board is None:
        board = [0] * n
    if solutions is None:
        solutions = []

    if row == n:
        solutions.append(board[:])
        return solutions

    for col in range(1, n + 1):
        if is_safe(board, row, col):
            board[row] = col
            find_all_solutions(row + 1, n, board, solutions)
            board[row] = 0

    return solutions

def format_solution(board):
    """Format solution as c(X,Y) pairs"""
```

```
    result = []
    for x, y in enumerate(board, 1):
        result.append(f"c({x},{y})")
    return "[" + ", ".join(result) + "]"

def print_board(board):
    """Print a visual representation of the board"""
    n = len(board)
    for i in range(n):
        row = ""
        for j in range(n):
            if board[i] == j + 1:
                row += "  "
            else:
                row += ". "
        print(row)
    print()

# Solve for first solution
print("=== First Solution ===")
board = [0] * 8
if solve_queens(board, 0):
    print("Solution found:")
    print(format_solution(board))
    print("\nBoard visualization:")
    print_board(board)

# Find all solutions
print("=== All Solutions ===")
all_solutions = find_all_solutions(0, 8)
print(f"Total solutions: {len(all_solutions)}\n")

for i, solution in enumerate(all_solutions, 1):
    print(f"Solution {i}: {format_solution(solution)}")

print(f"\n(There are {len(all_solutions)} distinct solutions to the
   8-Queens problem)")
```

Assignment (2)

Observe the result of the above program and discuss the result. Test the goal by placing a few queens explicitly.

(Try goals such as `solution([c(1,1),c(2,B),c(3,C),c(4,8),c(5,E),c(6,F),c(7,G),c(8,H)])` etc.)

Assignment (3)

Try to solve the above problem using C, C++, .NET or Java.

Assignment (4)

Create a DLL file using prolog and use it in any of the available languages python or .NET to depict the solution. Make it interactive. (You may want to set a queen in the fifth row of the first column)

https://claude.ai/public/artifacts/56ff3e10-a5be-4ab2-bd93-17aeaa4a1c12