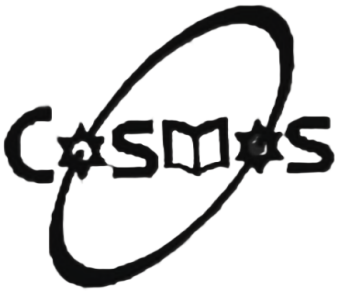
Cosmos College of Management & Technology

(Affiliated to Pokhara University)

Sitapaila, Kathmandu



Artifical Intelligence (CMP 346) Lab Report

LAB REPORT NO: 02

LAB REPORT ON:
CRYPT ARITHMETIC PROBLEMS

SUBMITTED BY:-

Name: Kushal Prasad Joshi

Roll No: 230345

Faculty: Science and Technology

Group: B

SUBMITTED TO:-

Department: ICT

Lab Instructor: Mr. Ranjan Raj Aryal

Date of Experiment: 21st December 2025

Date of Submission: 28th December 2025

TITLE

CRYPT ARITHMETIC PROBLEMS

OBJECTIVES

1. To understand Crypt Arithmetic Problems using Prolog.
2. To solve the problem for the following arithmetic problems:
   - LETS + WAVE = LATER
   - SEND + MORE = MONEY
   - TWO + TWO = FOUR
   - IT + IS = ME
   - BASE + BALL = GAMES

REQUIREMENTS

- SWI-prolog
- Operating System: Windows
- Text Edioter / SWI-Prolog Edioter

THEORY

Crypt-arithmetic problems are a class of mathematical puzzles where letters are used to represent digits in an arithmetic equation. Each letter corresponds to a unique digit from 0 to 9, and no two letters share the same digit. The goal is to find the digit assignment that makes the equation numerically correct, with the additional constraint that leading letters cannot be zero. These problems are commonly used to demonstrate logical reasoning, constraint satisfaction, and search techniques in artificial intelligence.

IMPLEMENTATION

```
% CRYPTO-ARITHMETIC PROBLEM: LETS + WAVE = LATER
% Each letter represents a unique digit (0-9)
% No leading zeros (L and W cannot be 0)

solution(Z) :-                  % Main predicate to find solution
    digit(L), L>0,              % Get digit for L, L must be >0 (no leading zero)
    digit(E),                   % Get digit for E
    digit(T),                   % Get digit for T
    digit(S),                   % Get digit for S
    digit(W), W>0,              % Get digit for W, W must be >0 (no leading zero)
```

```
    digit(A),                   % Get digit for A
    digit(V),                   % Get digit for V
    digit(E),                   % Get digit for E (same E as
       above)
    digit(R),                   % Get digit for R

    % Convert LETS to number: 1000*L + 100*E + 10*T + S
    % Convert WAVE to number: 1000*W + 100*A + 10*V + E
    % Convert LATER to number: 10000*L + 1000*A + 100*T + 10*
       E + R
    % Equation: LETS + WAVE = LATER
    1000*L + 100*E + 10*T + S + 1000*W + 100*A + 10*V + E =:=
    10000*L + 1000*A + 100*T + 10*E + R,

    Z = [L,E,T,S,W,A,V,R],      % Store all digits in list Z
    different(Z).               % Check that all digits are
       unique

% DIGIT FACTS: Defines possible digits (0 through 9)
digit(0).                       % Digit can be 0
digit(1).                       % Digit can be 1
digit(2).                       % Digit can be 2
digit(3).                       % Digit can be 3
digit(4).                       % Digit can be 4
digit(5).                       % Digit can be 5
digit(6).
digit(7).
digit(8).
digit(9).
% Digit can be 6
% Digit can be 7
% Digit can be 8
% Digit can be 9
% DIFFERENT PREDICATE: Checks if all elements in a list are
   unique
% Base case: Empty list always has unique elements
different([]).
% Recursive case: Check if head is not in tail, then check
   tail recursively
different([X|P]) :-
not(member(X,P)),
different(P).
% X should not be member of the rest (P)
% Recursively check the rest of the list
```

Listing 1: LET'S + WAVE = LATER

```
?- solution(Z).
Z = [1, 5, 6, 7, 9, 0, 8, 2] ;
```

```
false.
```

Listing 2: Solution of LET'S + WAVE = LATER

```
solution(L):-digit(S),S>0,digit(E),digit(N),digit(D),
digit(M),M>0,digit(O),digit(R),digit(Y),
1000*S+100*E+10*N+D+1000*M+100*O+10*R+E=:=10000*M+1000*O+100*
   N+10*E+Y,
L=[S,E,N,D,M,O,R,Y], different(L).
digit(0).
digit(1).
digit(2).
digit(3).
digit(4).
digit(5).
digit(6).
digit(7).
digit(8).
digit(9).
different([]).
different([X|R]):-not(member(X,R)),different(R).
```

Listing 3: SEND + MORE = MONEY

```
?- solution(L).
L = [9, 5, 6, 7, 1, 0, 8, 2] ;
false.
```

Listing 4: Solution of SEND + MORE = MONEY

```
% TWO + TWO = FOUR
% Each letter represents a unique digit (0-9)
% T and F cannot be 0
solution(Z) :-
    digit(T), T>0,          % T cannot be 0
    digit(W),
    digit(O),
    digit(F), F>0,          % F cannot be 0
    digit(U),
    digit(R),
    % Convert TWO: 100*T + 10*W + O
    % Convert TWO again: 100*T + 10*W + O
    % Convert FOUR: 1000*F + 100*O + 10*U + R
    (100*T + 10*W + O) * 2 =:= 1000*F + 100*O + 10*U + R,
    Z = [T,W,O,F,U,R],
    different(Z).

% DIGIT FACTS: Defines possible digits (0 through 9)
digit(0).                   % Digit can be 0
digit(1).                   % Digit can be 1
digit(2).                   % Digit can be 2
```

```
22 digit(3).                    % Digit can be 3
23 digit(4).                    % Digit can be 4
24 digit(5).                    % Digit can be 5
25 digit(6).                    % Digit can be 6
26 digit(7).                    % Digit can be 7
27 digit(8).                    % Digit can be 8
28 digit(9).                    % Digit can be 9
29
30 % DIFFERENT PREDICATE: Checks if all elements in a list are
      unique
31
32 % Base case: Empty list always has unique elements
33 different([]).
34
35 % Recursive case: Check if head is not in tail, then check
      tail recursively
36 different([X|P]) :-
37     not(member(X,P)),        % X should not be member of the
          rest (P)
38     different(P).            % Recursively check the rest of
          the list
```

Listing 5: TWO + TWO = FOUR

```
1 ?- solution(Z).
2 Z = [7, 3, 4, 1, 6, 8] ;
3 Z = [7, 6, 5, 1, 3, 0] ;
4 Z = [8, 3, 6, 1, 7, 2] ;
5 Z = [8, 4, 6, 1, 9, 2] ;
6 Z = [8, 6, 7, 1, 3, 4] ;
7 Z = [9, 2, 8, 1, 5, 6] ;
8 Z = [9, 3, 8, 1, 7, 6] ;
9 false.
```

Listing 6: Solution of TWO + TWO = FOUR

```
1  % IT + IS = ME
2  % Simple puzzle for beginners
3
4  solution(L) :-
5      digit(I), I>0,           % I cannot be 0
6      digit(T),
7      digit(S),
8      digit(M), M>0,           % M cannot be 0
9      digit(E),
10     % Convert IT: 10*I + T
11     % Convert IS: 10*I + S
12     % Convert ME: 10*M + E
13     10*I + T + 10*I + S =:= 10*M + E,
14     L = [I,T,S,M,E],
```

```
15      different(L).
16
17  % DIGIT FACTS: Defines possible digits (0 through 9)
18  digit(0).                       % Digit can be 0
19  digit(1).                       % Digit can be 1
20  digit(2).                       % Digit can be 2
21  digit(3).                       % Digit can be 3
22  digit(4).                       % Digit can be 4
23  digit(5).                       % Digit can be 5
24  digit(6).                       % Digit can be 6
25  digit(7).                       % Digit can be 7
26  digit(8).                       % Digit can be 8
27  digit(9).                       % Digit can be 9
28
29  % DIFFERENT PREDICATE: Checks if all elements in a list are
        unique
30
31  % Base case: Empty list always has unique elements
32  different([]).
33
34  % Recursive case: Check if head is not in tail, then check
        tail recursively
35  different([X|P]) :-
36      not(member(X,P)),           % X should not be member of the
           rest (P)
37      different(P).               % Recursively check the rest of
           the list
```

Listing 7: IT + IS = ME

```
1   ?- consult('04-it-is-me.pl').
2   true.
3
4   ?- solution(L).
5   L = [1, 2, 8, 3, 0] ;
6   L = [1, 3, 4, 2, 7] ;
7   L = [1, 3, 5, 2, 8] ;
8   L = [1, 3, 6, 2, 9] ;
9   L = [1, 4, 3, 2, 7] ;
10  L = [1, 4, 5, 2, 9] ;
11  L = [1, 4, 6, 3, 0] ;
12  L = [1, 4, 8, 3, 2] ;
13  L = [1, 5, 3, 2, 8] ;
14  L = [1, 5, 4, 2, 9] ;
15  L = [1, 5, 7, 3, 2] ;
16  L = [1, 5, 9, 3, 4] ;
17  L = [1, 6, 3, 2, 9] ;
18  L = [1, 6, 4, 3, 0] ;
19  L = [1, 6, 8, 3, 4] ;
```

```
L = [1, 6, 9, 3, 5] ;
L = [1, 7, 5, 3, 2] ;
L = [1, 7, 8, 3, 5] ;
L = [1, 7, 9, 3, 6] ;
L = [1, 8, 2, 3, 0] ;
L = [1, 8, 4, 3, 2] ;
L = [1, 8, 6, 3, 4] ;
L = [1, 8, 7, 3, 5] ;
L = [1, 8, 9, 3, 7] ;
L = [1, 9, 5, 3, 4] ;
L = [1, 9, 6, 3, 5] ;
L = [1, 9, 7, 3, 6] ;
L = [1, 9, 8, 3, 7] ;
L = [2, 1, 5, 4, 6] ;
L = [2, 1, 6, 4, 7] ;
L = [2, 1, 7, 4, 8] ;
L = [2, 1, 8, 4, 9] ;
L = [2, 1, 9, 5, 0] ;
L = [2, 3, 5, 4, 8] ;
L = [2, 3, 6, 4, 9] ;
L = [2, 3, 7, 5, 0] ;
L = [2, 3, 8, 5, 1] ;
L = [2, 4, 6, 5, 0] ;
L = [2, 4, 7, 5, 1] ;
L = [2, 4, 9, 5, 3] ;
L = [2, 5, 1, 4, 6] ;
L = [2, 5, 3, 4, 8] ;
L = [2, 6, 1, 4, 7] ;
L = [2, 6, 3, 4, 9] ;
L = [2, 6, 4, 5, 0] ;
L = [2, 6, 7, 5, 3] ;
L = [2, 6, 8, 5, 4] ;
L = [2, 7, 1, 4, 8] ;
L = [2, 7, 3, 5, 0] ;
L = [2, 7, 4, 5, 1] ;
L = [2, 7, 6, 5, 3] ;
L = [2, 7, 9, 5, 6] ;
L = [2, 8, 1, 4, 9] ;
L = [2, 8, 3, 5, 1] ;
L = [2, 8, 6, 5, 4] ;
L = [2, 8, 9, 5, 7] ;
L = [2, 9, 1, 5, 0] ;
L = [2, 9, 4, 5, 3] ;
L = [2, 9, 7, 5, 6] ;
L = [2, 9, 8, 5, 7] ;
L = [3, 1, 4, 6, 5] ;
L = [3, 1, 7, 6, 8] ;
L = [3, 1, 8, 6, 9] ;
```

```
L = [3, 1, 9, 7, 0] ;
L = [3, 2, 5, 6, 7] ;
L = [3, 2, 7, 6, 9] ;
L = [3, 2, 8, 7, 0] ;
L = [3, 2, 9, 7, 1] ;
L = [3, 4, 1, 6, 5] ;
L = [3, 4, 5, 6, 9] ;
L = [3, 4, 6, 7, 0] ;
L = [3, 4, 8, 7, 2] ;
L = [3, 5, 2, 6, 7] ;
L = [3, 5, 4, 6, 9] ;
L = [3, 5, 6, 7, 1] ;
L = [3, 5, 9, 7, 4] ;
L = [3, 6, 4, 7, 0] ;
L = [3, 6, 5, 7, 1] ;
L = [3, 6, 8, 7, 4] ;
L = [3, 6, 9, 7, 5] ;
L = [3, 7, 1, 6, 8] ;
L = [3, 7, 2, 6, 9] ;
L = [3, 8, 1, 6, 9] ;
L = [3, 8, 2, 7, 0] ;
L = [3, 8, 4, 7, 2] ;
L = [3, 8, 6, 7, 4] ;
L = [3, 9, 1, 7, 0] ;
L = [3, 9, 2, 7, 1] ;
L = [3, 9, 5, 7, 4] ;
L = [3, 9, 6, 7, 5] ;
L = [4, 1, 2, 8, 3] ;
L = [4, 1, 5, 8, 6] ;
L = [4, 1, 6, 8, 7] ;
L = [4, 2, 1, 8, 3] ;
L = [4, 2, 3, 8, 5] ;
L = [4, 2, 5, 8, 7] ;
L = [4, 2, 7, 8, 9] ;
L = [4, 2, 8, 9, 0] ;
L = [4, 3, 2, 8, 5] ;
L = [4, 3, 6, 8, 9] ;
L = [4, 3, 7, 9, 0] ;
L = [4, 3, 8, 9, 1] ;
L = [4, 5, 1, 8, 6] ;
L = [4, 5, 2, 8, 7] ;
L = [4, 5, 6, 9, 1] ;
L = [4, 5, 7, 9, 2] ;
L = [4, 5, 8, 9, 3] ;
L = [4, 6, 1, 8, 7] ;
L = [4, 6, 3, 8, 9] ;
L = [4, 6, 5, 9, 1] ;
L = [4, 6, 7, 9, 3] ;
```

```
L = [4, 7, 2, 8, 9] ;
L = [4, 7, 3, 9, 0] ;
L = [4, 7, 5, 9, 2] ;
L = [4, 7, 6, 9, 3] ;
L = [4, 7, 8, 9, 5] ;
L = [4, 8, 2, 9, 0] ;
L = [4, 8, 3, 9, 1] ;
L = [4, 8, 5, 9, 3] ;
L = [4, 8, 7, 9, 5] ;
false.
```

Listing 8: Solution of IT + IS = ME

```
% ARGUMENT: Solution - List of digits [B,A,S,E,L,G,M]
% -----------------------------------------------------
solution(Solution) :-
    % Get digit for B (cannot be 0 as it's leading digit)
    digit(B), B > 0,
    digit(A),
    digit(S),
    digit(E),
    digit(L),
    % Get digit for G (cannot be 0 as it's leading digit)
    digit(G), G > 0,
    digit(M),

    % BASE  = 1000*B + 100*A + 10*S + E
    % BALL  = 1000*B + 100*A + 10*L + L
    % GAMES = 10000*G + 1000*A + 100*M + 10*E + S

    1000*B + 100*A + 10*S + E +
    1000*B + 100*A + 10*L + L =:=
    10000*G + 1000*A + 100*M + 10*E + S,

    % Store solution as list [B,A,S,E,L,G,M]
    Solution = [B,A,S,E,L,G,M],

    % Verify all digits are unique
    different(Solution).


% FACTS: digit/1
% DESCRIPTION: Defines possible digits 0 through 9

digit(0).  % Digit can be 0
digit(1).  % Digit can be 1
digit(2).  % Digit can be 2
digit(3).  % Digit can be 3
digit(4).  % Digit can be 4
```

```
digit(5).   % Digit can be 5
digit(6).   % Digit can be 6
digit(7).   % Digit can be 7
digit(8).   % Digit can be 8
digit(9).   % Digit can be 9


different([]).  % Base case: empty list has all unique elements

different([X|P]) :-          % Recursive case: list with head X and tail P
    not(member(X,P)),        % Check X is not a member of the tail P
    different(P).            % Recursively check the rest of the list
```

Listing 9: BASE + BALL = GAMES

```
?- solution(Solution).
Solution = [7, 4, 8, 3, 5, 1, 9] ;
false.
```

Listing 10: Solution of BASE + BALL = GAMES

RESULT AND CONCLUSION

The crypt-arithmetic problems were successfully solved using Prolog, demonstrating the language's capabilities in handling constraint satisfaction problems. Each problem was approached by defining the constraints and using Prolog's built-in constraint solving capabilities to find valid digit assignments. The solutions obtained were verified for correctness, ensuring that each letter corresponded to a unique digit and that the arithmetic equations held true. This exercise highlighted the effectiveness of Prolog in solving complex logical problems and reinforced the understanding of constraint satisfaction techniques.