1 Prolog Code for Crypto-Arithmetic Problem

1.1

$$WIN + WIN = LOSE$$

```
DOMAINS
int_list = integer*

PREDICATES
solution(int_list)
member(integer, int_list)

CLAUSES
solution([]).
solution([W, I, N, L, O, S, E]) :-
    C4 = 1,
    member(C1, [0, 1]),
    member(C2, [0, 1]),
    member(C3, [0, 1]),
    member(W, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(I, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(N, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(L, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(O, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(S, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    member(E, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
    W <> I, W <> N, W <> L, W <> O, W <> S, W <> E,
    I <> N, I <> L, I <> O, I <> S, I <> E,
    N <> L, N <> O, N <> S, N <> E,
    L <> O, L <> S, L <> E,
    O <> S, O <> E,
    S <> E,
    N + N = E + 10 * C1,
    I + I + C1 = S + 10 * C2,
    W + W + C2 = O + 10 * C3,
    W + W + C3 = L + 10 * C4,
    L = C4.

member(X, [X | _]).
member(X, [_ | Z]) :-
    member(X, Z).

GOAL
solution([W, I, N, L, O, S, E]).
```

Listing 1: Prolog Code for Solution Finding

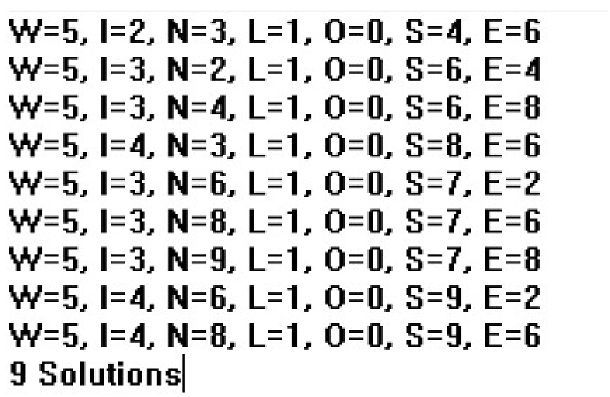

Figure 1: Output

1.2

$$TWO + TWO = FOUR$$

```

DOMAINS
int_list = integer*
PREDICATES
solution(int_list)
member(integer, int_list)
CLAUSES
solution([]).
solution([O, N, E, T, W, F, U, R]) :-
C3 = 1, member(C1, [0, 1]), member(C2, [0, 1]),
Digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
member(O, Digits), member(N, Digits), member(E, Digits),
member(T, Digits), member(W, Digits), member(F, Digits),
member(U, Digits), member(R, Digits),
O <> N, O <> E, O <> T, O <> W, O <> F, O <> U, O <> R,
N <> E, N <> T, N <> W, N <> F, N <> U, N <> R,
E <> T, E <> W, E <> F, E <> U, E <> R,
T <> W, T <> F, T <> U, T <> R,
W <> F, W <> U, W <> R,
F <> U, F <> R,
R <> U,
E + E + O= R + 10*C1,
N + N + W + C1 = U + 10*C2,
O + O + T + C2 = O + 10*C3,
F = C3.
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
GOAL
solution([O, N, E, T, W, F, U, R]).
```

Listing 2: Prolog Code for Solution Finding

```
O=2, N=6, E=4, T=7, W=5, F=1, U=8, R=0
O=3, N=2, E=8, T=6, W=5, F=1, U=0, R=9
O=3, N=7, E=8, T=6, W=0, F=1, U=5, R=9
O=4, N=3, E=7, T=5, W=9, F=1, U=6, R=8
O=4, N=6, E=7, T=5, W=0, F=1, U=3, R=8
O=4, N=8, E=3, T=5, W=2, F=1, U=9, R=0
O=5, N=2, E=6, T=4, W=8, F=1, U=3, R=7
O=5, N=2, E=7, T=4, W=8, F=1, U=3, R=9
O=5, N=6, E=7, T=4, W=0, F=1, U=3, R=9
O=5, N=8, E=6, T=4, W=2, F=1, U=9, R=7
O=6, N=4, E=2, T=3, W=8, F=1, U=7, R=0
O=6, N=4, E=2, T=3, W=9, F=1, U=8, R=0
O=6, N=5, E=2, T=3, W=7, F=1, U=8, R=0
O=6, N=5, E=2, T=3, W=8, F=1, U=9, R=0
O=6, N=7, E=2, T=3, W=4, F=1, U=9, R=0
O=7, N=3, E=4, T=2, W=9, F=1, U=6, R=5
O=7, N=3, E=6, T=2, W=8, F=1, U=5, R=9
O=7, N=5, E=6, T=2, W=3, F=1, U=4, R=9
O=7, N=6, E=4, T=2, W=0, F=1, U=3, R=5
O=9, N=2, E=4, T=0, W=8, F=1, U=3, R=7
O=9, N=3, E=4, T=0, W=5, F=1, U=2, R=7
O=9, N=3, E=4, T=0, W=8, F=1, U=5, R=7
O=9, N=4, E=2, T=0, W=6, F=1, U=5, R=3
O=9, N=4, E=2, T=0, W=7, F=1, U=6, R=3
O=9, N=4, E=2, T=0, W=8, F=1, U=7, R=3
O=9, N=4, E=3, T=0, W=7, F=1, U=6, R=5
O=9, N=4, E=3, T=0, W=8, F=1, U=7, R=5
O=9, N=5, E=2, T=0, W=6, F=1, U=7, R=3
O=9, N=5, E=2, T=0, W=7, F=1, U=8, R=3
O=9, N=5, E=4, T=0, W=2, F=1, U=3, R=7
O=9, N=6, E=2, T=0, W=4, F=1, U=7, R=3
O=9, N=6, E=2, T=0, W=5, F=1, U=8, R=3
O=9, N=6, E=3, T=0, W=4, F=1, U=7, R=5
O=9, N=6, E=4, T=0, W=2, F=1, U=5, R=7
O=9, N=6, E=4, T=0, W=5, F=1, U=8, R=7
B9 Solutions
```

Figure 2: Output

1.3

$$LOGIC + LOGIC = PROLOG$$

```
DOMAINS
int_list = integer*
PREDICATES
solution(int_list)
member(integer, int_list)
CLAUSES
solution([]).
solution([L, O, G, I, C, P, R]) :-
C5 = 1, member(C1, [0, 1]), member(C2, [0, 1]),
member(C3, [0, 1]), member(C4, [0, 1]),
Digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
member(L, Digits), member(O, Digits), member(G, Digits),
member(I, Digits), member(C, Digits), member(P, Digits),
member(R, Digits),
L <> O, L <> G, L <> I, L <> C, L <> P, L <> R,
O <> G, O <> I, O <> C, O <> P, O <> R,
G <> I, G <> C, G <> P, G <> R,
I <> C, I <> P, I <> R,
C <> P, C <> R,
P <> R,
C + C = G + 10*C1,
I + I + C1 = O + 10*C2,
G + G + C2 = L + 10*C3,
O + O + C3 = O + 10*C4,
L + L + C4 = R + 10*C5,
P = C5.
member(X, [X|T]).
member(X, [H|T]) :- member(X, T).
GOAL
solution([L, O, G, I, C, P, R]).
```

Listing 3: Prolog Code

```
L=9, O=0, G=4, I=5, C=2, P=1, R=8
1 Solution
```

Figure 3: Output

2 N-Queens Problems

2.1 Prolog implementation

```
DOMAINS
cell = c(integer, integer)
list = cell*
int_list = integer*
PREDICATES
solution(list)
member(integer, int_list)
noattack(cell, list)
CLAUSES
solution([]).
solution([c(X, Y)|Others]) :-
solution(Others),
member(Y, [1, 2, 3, 4, 5, 6, 7, 8]),
noattack(c(X, Y), Others).
noattack(_, []).
noattack(c(X, Y), [c(X1, Y1)|Others]) :-
Y <> Y1, Y1 - Y <> X1 - X, Y1 - Y <> X - X1,
noattack(c(X, Y), Others).
member(X, [X|_]).
member(X, [_|Z]) :- member(X, Z).
```

```
GOAL
solution([c(1, A), c(2, B), c(3, C), c(4, D),c(5, E), c(6, F), c(7, G), c(8, H)]).
```

Listing 4: Prolog Code for N queens problem

```
A=3, B=6, C=8, D=2, E=4, F=1, G=7, H=5
A=6, B=3, C=1, D=8, E=4, F=2, G=7, H=5
A=8, B=4, C=1, D=3, E=6, F=2, G=7, H=5
A=4, B=8, C=1, D=3, E=6, F=2, G=7, H=5
A=2, B=6, C=8, D=3, E=1, F=4, G=7, H=5
A=7, B=2, C=6, D=3, E=1, F=4, G=8, H=5
A=3, B=6, C=2, D=7, E=1, F=4, G=8, H=5
A=4, B=7, C=3, D=8, E=2, F=5, G=1, H=6
A=4, B=8, C=5, D=3, E=1, F=7, G=2, H=6
A=3, B=5, C=8, D=4, E=1, F=7, G=2, H=6
A=4, B=2, C=8, D=5, E=7, F=1, G=3, H=6
A=5, B=7, C=2, D=4, E=8, F=1, G=3, H=6
A=7, B=4, C=2, D=5, E=8, F=1, G=3, H=6
A=8, B=2, C=4, D=1, E=7, F=5, G=3, H=6
A=7, B=2, C=4, D=1, E=8, F=5, G=3, H=6
A=5, B=1, C=8, D=4, E=2, F=7, G=3, H=6
A=4, B=1, C=5, D=8, E=2, F=7, G=3, H=6
A=5, B=2, C=8, D=1, E=4, F=7, G=3, H=6
A=3, B=7, C=2, D=8, E=5, F=1, G=4, H=6
A=3, B=1, C=7, D=5, E=8, F=2, G=4, H=6
A=8, B=2, C=5, D=3, E=1, F=7, G=4, H=6
A=3, B=5, C=2, D=8, E=1, F=7, G=4, H=6
A=3, B=5, C=7, D=1, E=4, F=2, G=8, H=6
A=5, B=2, C=4, D=6, E=8, F=3, G=1, H=7
A=6, B=3, C=5, D=8, E=1, F=4, G=2, H=7
A=5, B=8, C=4, D=1, E=3, F=6, G=2, H=7
A=4, B=2, C=5, D=8, E=6, F=1, G=3, H=7
A=4, B=6, C=1, D=5, E=2, F=8, G=3, H=7
A=6, B=3, C=1, D=8, E=5, F=2, G=4, H=7
A=5, B=3, C=1, D=6, E=8, F=2, G=4, H=7
A=4, B=2, C=8, D=6, E=1, F=3, G=5, H=7
A=6, B=3, C=5, D=7, E=1, F=4, G=2, H=8
A=6, B=4, C=7, D=1, E=3, F=5, G=2, H=8
A=4, B=7, C=5, D=2, E=6, F=1, G=3, H=8
A=5, B=7, C=2, D=6, E=3, F=1, G=4, H=8
92 Solutions
```

Figure 4: Output

2.2 Classical Implementation

```
#include <iostream>
#include <vector>
class Queens {
public:
int solutionCount, boardSize;
std::vector<int> cells;
public:
Queens(int b) {
boardSize = b;
solutionCount = 0;
cells.resize(boardSize);
for(int i = 0; i < boardSize; i++) {
cells[i] = i;
}
}
void solve(int row = 0) {
if(row >= boardSize) {
printBoard();
std::cout << ++solutionCount << "\n\n";
}
for(int col = 0; col < boardSize; col++) {
if(noAttack(row, col)) {
cells[row] = col;
solve(row+1);
}
}
}
bool noAttack(int row, int col) {
for(int i = 0; i < row; i++) {
if(cells[i] == col or
abs(i-row) == abs(cells[i]-col)) return false;
}
return true;
}
void printBoard() {
int i = 0;
for(int x = 0; x < boardSize; x++) {
for(int y = 0; y < boardSize; y++) {
if(i < boardSize and x == i and y == cells[i]) {
std::cout << "x ";
i++;
} else {
std::cout << ". ";
}
}
std::cout << std::endl;
}
}
};
int main() {
int boardSize;
std::cout << "Enter the boards size: ";
std::cin >> boardSize;
Queens solver(boardSize);
solver.solve();
std::cout << solver.solutionCount << " solutions found.\n";
return 0;
}
```

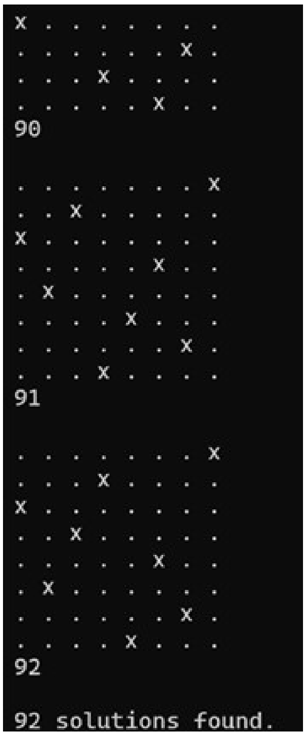

Figure 5: Output

3 Classical - Search Problems

3.1 Water-jug problem

```
DOMAINS
state = c(integer, integer)
path = state*
path_list = path*
PREDICATES
enqueue(path_list, path, path_list)
member(state, path)
enqueueFF(path_list, path, path_list, path, path, integer, integer)
enqueueFS(path_list, path, path_list, path, path, integer, integer)
enqueueEF(path_list, path, path_list, path, path, integer, integer)
enqueueES(path_list, path, path_list, path, path, integer, integer)
enqueueFTS(path_list, path, path_list, path, path, integer, integer)
enqueueSTF(path_list, path, path_list, path, path, integer, integer)
fullFirst(state, state, integer, integer)
fullSecond(state, state, integer, integer)
emptyFirst(state, state, integer, integer)
emptySecond(state, state, integer, integer)
firstToSecond(state, state, integer, integer)
secondToFirst(state, state, integer, integer)
bfs(path_list, integer, path, path, integer, integer)
start(integer, integer, integer)
CLAUSES
start(Cx, Cy, G) :- bfs([[c(0, 0)]], G, [c(0, 0)], V, Cx, Cy), !.
bfs([], _, _, _, _, _) :- write("Search Exhausted"), nl, !.
bfs([[c(X, Y)|R1]|R2], G, V, V, _, _) :- G = X + Y, write("Found: "),
P = [c(X, Y)|R1], reverse(P, [], P1), write(P1), nl, !.
bfs([P|R], G, V, W, Cx, Cy) :- enqueueFF(R, P, Q1, V, V1, Cx, Cy),
enqueueFS(Q1, P, Q2, V1, V2, Cx, Cy),
enqueueEF(Q2, P, Q3, V2, V3, Cx, Cy),
enqueueES(Q3, P, Q4, V3, V4, Cx, Cy),
enqueueFTS(Q4, P, Q5, V4, V5, Cx, Cy),
enqueueSTF(Q5, P, Q, V5, W, Cx, Cy),
bfs(Q, G, W, X, Cx, Cy), !.
enqueueFF(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- fullFirst(N, S, Cx, Cy),
not(member(S, V1)),
```

```
enqueue(Q1, [S, N|R], Q2),
V2 = [S|V1], !.
enqueueFF(Q, _, Q, V, V, _, _).
fullFirst(c(X, Y), c(Cx, Y), Cx, _).
enqueueFS(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- fullSecond(N, S, Cx, Cy),
not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2),
V2 = [S|V1], !.
enqueueFS(Q, _, Q, V, V, _, _).
fullSecond(c(X, Y), c(X, Cy), _, Cy).
enqueueEF(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- emptyFirst(N, S, Cx, Cy),
not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2),
V2 = [S|V1], !.
enqueueEF(Q, _, Q, V, V, _, _).
emptyFirst(c(X, Y), c(0, Y), _, _).
enqueueES(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- emptySecond(N, S, Cx,
Cy),
not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2),
V2 = [S|V1], !.
enqueueES(Q, _, Q, V, V, _, _).
emptySecond(c(X, Y), c(X, 0), _, _).
enqueueFTS(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- firstToSecond(N, S, Cx,
Cy),
not(member(S, V1)), enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueFTS(Q, _, Q, V, V, _, _).
firstToSecond(c(X, Y), c(A, B), Cx, Cy) :- Rem = Cy - Y, Rem >= X, A = 0,
B = Y + X, !.
firstToSecond(c(X, Y), c(A, B), Cx, Cy) :- Rem = Cy - Y, Rem < X,
A = X - Rem, B = Cy, !.
enqueueSTF(Q1, [N|R], Q2, V1, V2, Cx, Cy) :- secondToFirst(N, S, Cx,
Cy),
not(member(S, V1)), enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueSTF(Q, _, Q, V, V, _, _).
secondToFirst(c(X, Y), c(A, B), Cx, Cy) :- Rem = Cx - X, Rem >= Y,
A = X + Y, B = 0, !.
secondToFirst(c(X, Y), c(A, B), Cx, Cy) :- Rem = Cx - X, Rem < Y, A = Cx,
B = Y - Rem, !.
enqueue([], Element, [Element]).
enqueue([Head|Tail], Element, [Head|NewTail]) :-
enqueue(Tail, Element, NewTail).
member(Node, [Node|Rest]).
member(Node, [_|Rest]) :- member(Node, Rest).
reverse([], L, L) :- !.
reverse([H|T], L, Z) :- reverse(T, [H|L], Z).
GOAL
start(3, 5, 7).
```

Listing 5: Prolog Code

```
Found: [c(0,0),c(0,5),c(3,2),c(0,2),c(2,0),c(2,5)]
yes
```

Figure 6: Output

3.2 Eight-Puzzle Problem

```
DOMAINS
state = integer*
path = state*
path_list = path*
PREDICATES
enqueue(path_list, path, path_list)
empty(path_list)
member(state, path)
findin(integer, state, integer)
swap_indices(state, integer, integer, state)
```

```
split_at(integer, state, state, state)
append(state, state, state)
enqueueUp(path_list, path, path_list, path, path)
enqueueDown(path_list, path, path_list, path, path)
enqueueLeft(path_list, path, path_list, path, path)
enqueueRight(path_list, path, path_list, path, path)
bfs(path_list, state, path, path)
up(state, state)
down(state, state)
right(state, state)
left(state, state)
start(state)
reverse(path, path, path)
print(path)
CLAUSES
start(S) :- bfs([[S]], [1, 2, 3, 4, 5, 6, 7, 8, 0], [S], V), !.
bfs([], _, _, _) :- write("Search Exhausted"), nl, !.
bfs([[G|R1]|R2], G, V, V) :-
write("Found: "), reverse([G|R1], [], P), nl, print(P), !.
bfs([P|R], G, V, W) :-
enqueueUp(R, P, Q1, V, V1),
enqueueDown(Q1, P, Q2, V1, V2),
enqueueRight(Q2, P, Q3, V2, V3),
enqueueLeft(Q3, P, Q, V3, W), bfs(Q, G, W, X).
enqueueUp(Q1, [N|Rest], Q2, V1, V2) :-
up(N, S), not(member(S, V1)), enqueue(Q1, [S|[N|Rest]], Q2),
V2 = [S|V1], !.
enqueueUp(Q1, _, Q1, V, V).
up(S, N) :-
findin(0, S, I), I > 2, J = I - 3, swap_indices(S, J, I, N), !.
up(S, S).
enqueueDown(Q1, [N|Rest], Q2, V1, V2) :-
down(N, S), not(member(S, V1)), enqueue(Q1, [S|[N|Rest]], Q2),
V2 = [S|V1], !.
enqueueDown(Q1, _, Q1, V, V).
down(S, N) :-
findin(0, S, I), I < 6, J = I + 3, swap_indices(S, I, J, N), !.
down(S, S).
enqueueLeft(Q1, [N|Rest], Q2, V1, V2) :-
left(N, S), not(member(S, V1)), enqueue(Q1, [S|[N|Rest]], Q2),
V2 = [S|V1], !.
enqueueLeft(Q1, _, Q1, V, V).
left(S, N) :-
findin(0, S, I), R = I mod 3, R > 0,
J = I - 1, swap_indices(S, J, I, N), !.
left(S, S).
enqueueRight(Q1, [N|Rest], Q2, V1, V2) :-
right(N, S), not(member(S, V1)),
enqueue(Q1, [S|[N|Rest]], Q2), V2 = [S|V1], !.
enqueueRight(Q1, _, Q1, V, V).
right(S, N) :- findin(0, S, I), R = I mod 3, R < 2,
J = I + 1, swap_indices(S, I, J, N), !.
right(S, S).
empty([]).
enqueue([], Element, [Element]).
enqueue([Head|Tail], Element, [Head|NewTail]) :-
enqueue(Tail, Element, NewTail).
member(Node, [Node|Rest]).
member(Node, [_|Rest]) :- member(Node, Rest).
findin(N, [N|_], 0).
findin(N, [_|Rest], Z) :- findin(N, Rest, L), Z = L + 1.
split_at(0, List, [], List).
split_at(N, [H|T], [H|Front], Rest) :-
N > 0,
N1 = N - 1,
split_at(N1, T, Front, Rest).
append([], L, L).
append([H|T], L, [H|Result]) :- append(T, L, Result).
swap_indices(List, I1, I2, Result) :-
I1 < I2,
split_at(I1, List, Before1, [Elem1|Middle1]),
SplitIndex2 = I2 - I1 - 1,
split_at(SplitIndex2, Middle1, Middle, [Elem2|After]),
```

```
append(Before1, [Elem2|Middle], TempList),
append(TempList, [Elem1|After], Result).
reverse([], L, L) :- !.
reverse([H|T], L, Z) :- reverse(T, [H|L], Z).
print([]) :- !.
print([[A, B, C, D, E, F, G, H, I]|R]) :-
write(A), write(" "), write(B), write(" "), write(C), nl,
write(D), write(" "), write(E), write(" "), write(F), nl,
write(G), write(" "), write(H), write(" "), write(I), nl,
nl, print(R).
GOAL
start([2, 3, 6, 7, 1, 4, 5, 0, 8]).
```

Listing 6: Prolog Code

```
Found:
2 3 6
7 1 4
5 0 8

2 3 6
7 1 4
0 5 8

2 3 6
0 1 4
7 5 8

2 3 6
1 0 4
7 5 8

2 3 6
1 4 0
7 5 8

2 3 0
1 4 6
7 5 8

2 0 3
1 4 6
7 5 8

0 2 3
1 4 6
7 5 8

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
```

Figure 7: Output

3.3 Missionaries and Cannibals

```
DOMAINS
state = c(integer, integer, integer, integer, integer, integer, integer)
path = state*
path_list = path*
PREDICATES
enqueue(path_list, path, path_list)
member(state, path)
enqueueMB(path_list, path, path_list, path, path)
moveBoat(state, state)
enqueueLB(path_list, path, path_list, path, path, integer, integer)
leftToBoat(state, state, integer, integer)
enqueueRB(path_list, path, path_list, path, path, integer, integer)
rightToBoat(state, state, integer, integer)
enqueueBL(path_list, path, path_list, path, path, integer, integer)
boatToLeft(state, state, integer, integer)
enqueueBR(path_list, path, path_list, path, path, integer, integer)
```

```
boatToRight(state, state, integer, integer)
bfs(path_list, state, path, path)
print(path)
reverse(path, path, path)
start
CLAUSES
start :-
bfs([[c(0, 0, 0, 0, 3, 3, 1)]], c(3, 3, 0, 0, 0, 0, _),
[c(0, 0, 0, 0, 3, 3, 1)], W), !.
bfs([], _, _, _) :- write("Search Exhausted"), nl, !.
bfs([[G|R1]|R2], G, V, V) :-
write("Found: "), reverse([G|R1], [], P),
print(P), nl, !.
bfs([P|R], G, V, W) :-
enqueueMB(R, P, Q1, V, V1),
enqueueLB(Q1, P, Q2, V1, V2, 0, 1),
enqueueLB(Q2, P, Q3, V2, V3, 0, 2),
enqueueLB(Q3, P, Q4, V3, V4, 1, 1),
enqueueLB(Q4, P, Q5, V4, V5, 1, 0),
enqueueLB(Q5, P, Q6, V5, V6, 2, 0),
enqueueRB(Q6, P, Q7, V6, V7, 0, 1),
enqueueRB(Q7, P, Q8, V7, V8, 0, 2),
enqueueRB(Q8, P, Q9, V8, V9, 1, 1),
enqueueRB(Q9, P, Q10, V9, V10, 1, 0),
enqueueRB(Q10, P, Q11, V10, V11, 2, 0),
enqueueBL(Q11, P, Q12, V11, V12, 0, 1),
enqueueBL(Q12, P, Q13, V12, V13, 0, 2),
enqueueBL(Q13, P, Q14, V13, V14, 1, 1),
enqueueBL(Q14, P, Q15, V14, V15, 1, 0),
enqueueBL(Q15, P, Q16, V15, V16, 2, 0),
enqueueBR(Q16, P, Q17, V16, V17, 0, 2),
enqueueBR(Q17, P, Q18, V17, V18, 0, 1),
enqueueBR(Q18, P, Q19, V18, V19, 1, 1),
enqueueBR(Q19, P, Q20, V19, V20, 1, 0),
enqueueBR(Q20, P, Q, V20, W, 2, 0),
bfs(Q, G, W, X).
enqueueMB(Q1, [N|R], Q2, V1, V2) :-
moveBoat(N, S), not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueMB(Q, _, Q, V, V).
moveBoat(c(A, B, 0, 0, E, F, G), c(A, B, 0, 0, E, F, G)) :- !.
moveBoat(c(Lc, Lm, Bc, Bm, Rc, Rm, 0), X) :-
Lc <= Lm, Rc + Bc <= Rm + Bm, X = c(Lc, Lm, Bc, Bm, Rc, Rm, 1), !.
moveBoat(c(Lc, Lm, Bc, 0, Rc, 0, 0), X) :-
Lc <= Lm, X = c(Lc, Lm, Bc, 0, Rc, 0, 1), !.
moveBoat(c(Lc, 0, Bc, Bm, Rc, Rm, 0), X) :-
Rc + Bc <= Rm + Bm, X = c(Lc, 0, Bc, Bm, Rc, Rm, 1), !.
moveBoat(c(Lc, Lm, Bc, Bm, Rc, Rm, 1), X) :-
Rc <= Rm, Lc + Bc <= Lm + Bm, X = c(Lc, Lm, Bc, Bm, Rc, Rm, 0), !.
moveBoat(c(Lc, 0, Bc, 0, Rc, Rm, 1), X) :-
Rc <= Rm, X = c(Lc, 0, Bc, 0, Rc, Rm, 0), !.
moveBoat(c(Lc, Lm, Bc, Bm, Rc, 0, 1), X) :-
Lc + Bc <= Lm + Bm, X = c(Lc, Lm, Bc, Bm, Rc, 0, 0), !.
moveBoat(S, S).
enqueueLB(Q1, [N|R], Q2, V1, V2, C, M) :-
leftToBoat(N, S, C, M), not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueLB(Q, _, Q, V, V, _, _).
leftToBoat(c(Lc, Lm, Bc, Bm, Rc, Rm, 0), X, C, M) :-
C <= Lc, M <= Lm, C + Bc + M + Bm <= 2,
NLc = Lc - C, NLm = Lm - M, NBc = Bc + C, NBm = Bm + M,
X = c(NLc, NLm, NBc, NBm, Rc, Rm, 0), !.
leftToBoat(S, S, _, _).
enqueueRB(Q1, [N|R], Q2, V1, V2, C, M) :-
rightToBoat(N, S, C, M), not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueRB(Q, _, Q, V, V, _, _).
rightToBoat(c(Lc, Lm, Bc, Bm, Rc, Rm, 1), X, C, M) :-
C <= Rc, M <= Rm, C + Bc + M + Bm <= 2,
NRc = Rc - C, NRm = Rm - M, NBc = Bc + C, NBm = Bm + M,
X = c(Lc, Lm, NBc, NBm, NRc, NRm, 1), !.
rightToBoat(S, S, _, _).
enqueueBL(Q1, [N|R], Q2, V1, V2, C, M) :-
```

```
boatToLeft(N, S, C, M), not(member(S, V1)),
enqueue(Q1, [S, N|R], Q2), V2 = [S|V1], !.
enqueueBL(Q, _, Q, V, V, _, _).
boatToLeft(c(Lc, Lm, Bc, Bm, Rc, Rm, 0), X, C, M) :- C <= Bc, M <= Bm,
NLc = Lc + C, NLm = Lm + M, NBc = Bc - C, NBm = Bm - M,
X = c(NLc, NLm, NBc, NBm, Rc, Rm, 0), !.
boatToLeft(S, S, _, _).
enqueueBR(Q1, [N|R], Q2, V1, V2, C, M) :- boatToRight(N, S, C, M),
not(member(S,enqueueBR(Q, _, Q, V, V, _, _).
boatToRight(c(Lc, Lm, Bc, Bm, Rc, Rm, 1), X, C, M) :- C <= Bc, M <= Bm,
NRc = Rc + C, NRm = Rm + M, NBc = Bc - C, NBm = Bm - M,
X = c(Lc, Lm, NBc, NBm, NRc, NRm, 1), !.
boatToRight(S, S, _, _).
enqueue([], Element, [Element]).
enqueue([Head|Tail], Element, [Head|NewTail]) :-
enqueue(Tail, Element, NewTail).
member(Node, [Node|Rest]).
member(Node, [_|Rest]) :- member(Node, Rest).
reverse([], L, L) :- !.
reverse([H|T], L, Z) :- reverse(T, [H|L], Z).
print([]) :- !.
print([c(Lc, Lm, Bc, Bm, Rc, Rm, B)|R]) :-
write("Left: "), write(Lc), write(" "), write(Lm), write(", Boat: "),
write(Bc), write(" "), write(Bm), write(" P: "),
write(B), write(", Right: "), write(Rc), write(" "),
write(Rm), nl, print(R).
GOAL
start.
```

Listing 7: Prolog Code

```
Found: Left: 0 0, Boat: 0 0 P: 1, Right: 3 3
Left: 0 0, Boat: 2 0 P: 1, Right: 1 3
Left: 0 0, Boat: 2 0 P: 0, Right: 1 3
Left: 1 0, Boat: 1 0 P: 0, Right: 1 3
Left: 1 0, Boat: 1 0 P: 1, Right: 1 3
Left: 1 0, Boat: 2 0 P: 1, Right: 0 3
Left: 1 0, Boat: 2 0 P: 0, Right: 0 3
Left: 2 0, Boat: 1 0 P: 0, Right: 0 3
Left: 2 0, Boat: 1 0 P: 1, Right: 0 3
Left: 2 0, Boat: 0 0 P: 1, Right: 1 3
Left: 2 0, Boat: 0 2 P: 1, Right: 1 1
Left: 2 0, Boat: 0 2 P: 0, Right: 1 1
Left: 2 1, Boat: 0 1 P: 0, Right: 1 1
Left: 1 1, Boat: 1 1 P: 0, Right: 1 1
Left: 1 1, Boat: 1 1 P: 1, Right: 1 1
Left: 1 1, Boat: 0 0 P: 1, Right: 2 2
Left: 1 1, Boat: 0 2 P: 1, Right: 2 0
Left: 1 1, Boat: 0 2 P: 0, Right: 2 0
Left: 1 3, Boat: 0 0 P: 0, Right: 2 0
Left: 0 3, Boat: 1 0 P: 0, Right: 2 0
Left: 0 3, Boat: 1 0 P: 1, Right: 2 0
Left: 0 3, Boat: 2 0 P: 1, Right: 1 0
Left: 0 3, Boat: 2 0 P: 0, Right: 1 0
Left: 1 3, Boat: 1 0 P: 0, Right: 1 0
Left: 1 3, Boat: 1 0 P: 1, Right: 1 0
Left: 1 3, Boat: 2 0 P: 1, Right: 0 0
Left: 1 3, Boat: 2 0 P: 0, Right: 0 0
Left: 3 3, Boat: 0 0 P: 0, Right: 0 0

yes
```

Figure 8: Output

4 Discussion

Constraint programming (CP) and classical search algorithms each have their own advantages and drawbacks. CP is particularly effective for problems like N-Queens, where constraints can be directly applied to the solution space. It works well for problems with complex constraints but may face scalability issues when dealing with large problem sizes. On the other hand, classical search algorithms such as Breadth-First Search (BFS) are efficient for problems with clear state spaces, like the Water-Jug problem. However, BFS can become inefficient with larger state spaces due to the rapid increase in possible configurations. While BFS guarantees the shortest path to a solution, it lacks the ability to use heuristics, making it less effective for complex problems. In general, CP is better suited for problems with heavy constraints, while classical search algorithms work best for problems that can be framed as state spaces.

5 Conclusion

In this lab, we explored the strengths and limitations of constraint programming and classical search algorithms. Constraint programming proved to be effective for combinatorial problems, while BFS offered clear solutions for simpler state-space problems. However, both methods require optimization or alternative approaches to overcome scalability issues in more complex scenarios.