Code:

```
from collections import deque
# State representation: (missionaries_on_left, cannibals_on_left, boat_position)
initial_state = (3, 3, 1)
goal_state = (0, 0, 0)

def is_valid_state(state):
    missionaries_left, cannibals_left, boat_position = state
    missionaries_right = 3 - missionaries_left
    cannibals_right = 3 - cannibals_left

    if (missionaries_left < 0 or missionaries_right < 0 or
        cannibals_left < 0 or cannibals_right < 0 or
        (missionaries_left > 0 and missionaries_left < cannibals_left) or
        (missionaries_right > 0 and missionaries_right < cannibals_right)):
        return False
    return True

def generate_next_states(state):
    next_states = []
    boat_position = state[2]

    for m in range(3):
        for c in range(3):
            if 0 < m + c <= 2:
                if boat_position == 1:  # Boat on the left
                    new_state = (state[0] - m, state[1] - c, 0)
                else:  # Boat on the right
                    new_state = (state[0] + m, state[1] + c, 1)

                if is_valid_state(new_state):
                    next_states.append(new_state)

    return next_states

def bfs(initial_state, goal_state):
    visited = set()
    queue = deque([(initial_state, [])])

    while queue:
        state, path = queue.popleft()

        if state == goal_state:
            return path
```

```
        if state not in visited:
            visited.add(state)
            for next_state in generate_next_states(state):
                queue.append((next_state, path + [state]))

    return None

def print_solution_path(path):
    for i, state in enumerate(path):
        print(f"Step {i + 1}: {state[0]} missionaries and {state[1]} cannibals on the left side, Boat
Position: {'Left' if state[2] == 1 else 'Right'}")

def main():
    path = bfs(initial_state, goal_state)

    if path is None:
        print("No solution found.")
    else:
        print("Solution found:")
        print_solution_path(path)

if __name__ == "__main__":
    main()
```

Output: