Vamshi Jandhyala

Solving the LinkedIn Queens Puzzle with Z3

✠

2024

The LinkedIn Queens puzzle places exactly one queen in each row, column, and colour region of an $n \times n$ board, with the additional constraint that no two queens may touch each other, even diagonally. The classical n-queens problem adds the colour-region constraint, which gives Z3 enough structure to find unique solutions in milliseconds.

Solution

The Python code below solves the puzzle using Z3 and visualises the result with matplotlib.

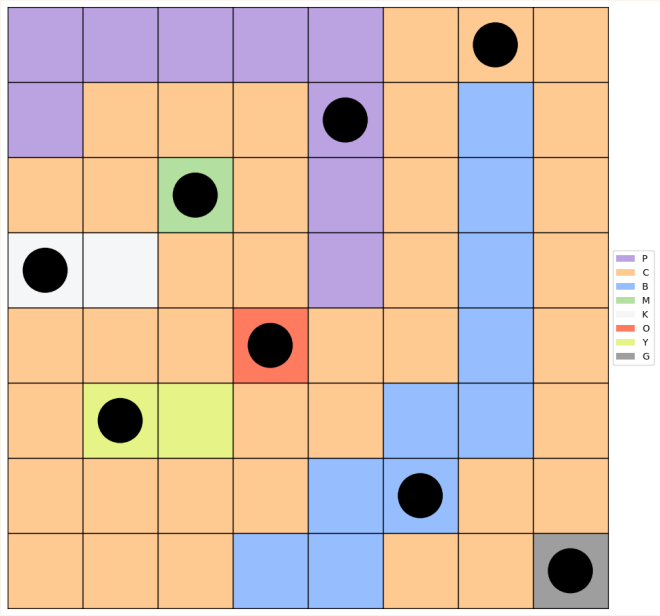

Figure 1: *
Puzzle 1.

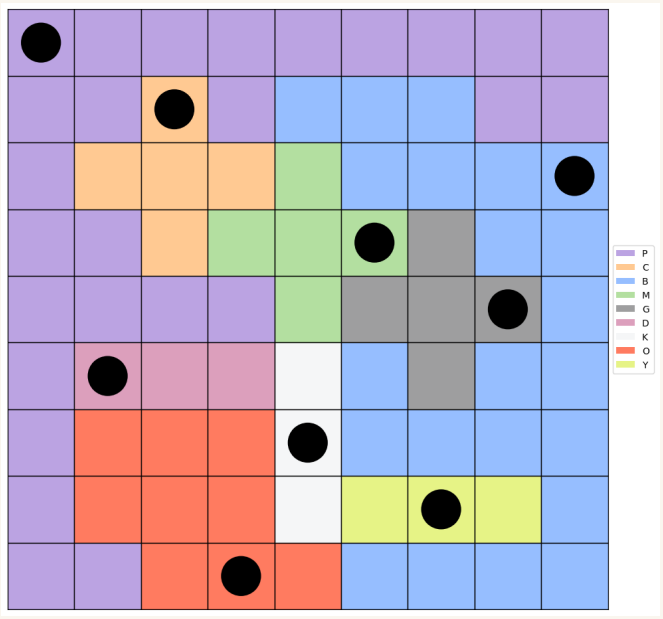

Figure 2: *
Puzzle 2.

Python code

```
from z3 import *
import matplotlib.pyplot as plt
from matplotlib.colors import to_rgba

def parse_color_grid(color_grid):
    n = len(color_grid)
    color_data = {}
    preplaced_queens = []
    for i, row in enumerate(color_grid):
        for j, cell in enumerate(row):
            color = cell[1] if cell.startswith('Q') else cell
            if color not in color_data:
                color_data[color] = []
            color_data[color].append((i, j))
            if cell.startswith('Q'):
                preplaced_queens.append((i, j))
    return color_data, preplaced_queens

def solve_n_queens_color(n, color_data, preplaced_queens):
    solver = Solver()
    grid = [[Int(f"cell_{i}_{j}") for j in range(n)] for i in range(n)]
    for row in grid:
        for cell in row:
            solver.add(Or(cell == 0, cell == 1))
    for i, j in preplaced_queens:
        solver.add(grid[i][j] == 1)
    for row in grid:
        solver.add(Sum(row) == 1)
    for j in range(n):
        solver.add(Sum([grid[i][j] for i in range(n)]) == 1)
```

```
    directions = [(-1, -1), (-1, 0), (-1, 1),
                  (0, -1),         (0, 1),
                  (1, -1), (1, 0), (1, 1)]
    for i in range(n):
        for j in range(n):
            for di, dj in directions:
                ni, nj = i + di, j + dj
                if 0 <= ni < n and 0 <= nj < n:
                    solver.add(Implies(grid[i][j] == 1, grid[ni][nj] == 0))
    for color, squares in color_data.items():
        solver.add(Sum([grid[i][j] for i, j in squares]) == 1)
    if solver.check() == sat:
        return solver.model(), grid
    return None, None

COLOR_MAP = {
    'P': '#bba3e2', 'Y': '#e6f386', 'O': '#ff7b60',
    'B': '#96beff', 'C': '#ffc992', 'G': '#9e9e9f',
    'K': '#f5f6f7', 'M': '#b3dfa0', 'D': '#dc9fbc',
}

def visualize_solution(model, grid, color_data, n):
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.set_aspect('equal')
    ax.set_xlim(0, n)
    ax.set_ylim(0, n)
    ax.set_xticks([])
    ax.set_yticks([])
    for i in range(n):
        for j in range(n):
            color = next(c for c, sq in color_data.items() if (i, j) in sq)
            ax.add_patch(plt.Rectangle((j, n - 1 - i), 1, 1,
                                       facecolor=COLOR_MAP.get(color, 'white'),
                                       edgecolor='black'))
            if model.evaluate(grid[i][j]) == 1:
                ax.add_patch(plt.Circle((j + 0.5, n - 0.5 - i),
                                        0.3, facecolor='black'))
    legend = [plt.Rectangle((0, 0), 1, 1, facecolor=COLOR_MAP[c])
              for c in color_data]
    ax.legend(legend, color_data.keys(),
              loc='center left', bbox_to_anchor=(1, 0.5))
    plt.tight_layout()
    plt.show()

def solve(color_grid):
    color_data, preplaced = parse_color_grid(color_grid)
    n = len(color_grid)
    model, grid = solve_n_queens_color(n, color_data, preplaced)
    if model:
        print("Solution found!")
        visualize_solution(model, grid, color_data, n)
    else:
        print("No solution found.")
```

Example puzzles

```
puzzle1 = [
```

```
    ['P','P','P','P','P','C','QC','C'],
    ['P','C','C','C','QP','C','B','C'],
    ['C','C','M','C','P','C','B','C'],
    ['K','K','C','C','P','C','B','C'],
    ['C','C','C','O','C','C','B','C'],
    ['C','Y','Y','C','C','B','B','C'],
    ['C','C','C','C','B','B','C','C'],
    ['C','C','C','B','B','C','C','G'],
]

puzzle2 = [
    ['P','P','P','P','P','P','P','P','P'],
    ['P','P','C','P','B','B','B','P','P'],
    ['P','C','C','C','M','B','B','B','B'],
    ['P','P','C','M','M','M','G','B','B'],
    ['P','P','P','P','M','G','G','G','B'],
    ['P','D','D','D','K','B','G','B','B'],
    ['P','O','O','O','K','B','B','B','B'],
    ['P','O','O','O','K','Y','Y','Y','B'],
    ['P','P','O','O','O','B','B','B','B'],
]

solve(puzzle2)
```