Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Pop Rares-Andrei
Group: 30434

Contents

1 Introduction

This document describes the MACE4 code used to define a puzzle based on the Latin Square and Futoshiki-like constraints. The puzzle consists of a 4x4 grid where numbers must be placed in such a way that satisfies both uniqueness in rows and columns (Latin Square) and inequality relationships between adjacent cells (Futoshiki).

2 General Settings

The following commands define the general settings for the puzzle solver.

```
set(arithmetic).
assign(domain_size,4).
assign(max_models,-1).
```

- `set(arithmetic)`: Enables the solver to use arithmetic constraints such as less than (¡), greater than (¿), and equality (=).
- `assign(domain_size,4)`: Sets the domain size to 4, meaning each variable can take values from the set $\{1, 2, 3, 4\}$.
- `assign(max_models,-1)`: Instructs the solver to find all possible solutions to the puzzle (no limit on the number of models).

3 Latin Square Constraints

The Latin Square constraints ensure that each number appears exactly once in every row and column.

```
formulas(latin_square).
  f(x, y1) = f(x, y2) -> y1 = y2.
  f(x1, y) = f(x2, y) -> x1 = x2.
end_of_list.
```

- `f(x, y1) = f(x, y2) -> y1 = y2`: If two cells in the same row have the same value, they must refer to the same column. This enforces the uniqueness of values in a row.
- `f(x1, y) = f(x2, y) -> x1 = x2`: If two cells in the same column have the same value, they must refer to the same row. This enforces the uniqueness of values in a column.

4 Futoshiki-like Constraints

The Futoshiki constraints define inequalities between adjacent cells. These constraints help add an extra layer of complexity to the puzzle.

```
formulas(futoshiki).
  f(0,0) < f(0,1).
  f(0,1) > f(0,2).
  f(1,0) < f(1,1).
  f(2,2) > f(2,3).
  f(3,0) > f(3,1).
  f(3,2) < f(3,3).
  f(0,3) > f(1,3).
  f(2,1) < f(2,2).
end_of_list.
```

- `f(0,0) < f(0,1)`: The value in the top-left cell must be less than the value in the top-middle cell.
- `f(0,1) > f(0,2)`: The value in the top-middle cell must be greater than the value in the top-right cell.
- `f(1,0) < f(1,1)`: The value in the second row, first column must be less than the value in the second row, second column.
- `f(2,2) > f(2,3)`: The value in the third row, third column must be greater than the value in the third row, fourth column.
- `f(3,0) > f(3,1)`: The value in the bottom row, first column must be greater than the value in the bottom row, second column.
- `f(3,2) < f(3,3)`: The value in the bottom row, third column must be less than the value in the bottom-right corner.
- `f(0,3) > f(1,3)`: The value in the top-right corner must be greater than the value in the second row, fourth column.
- `f(2,1) < f(2,2)`: The value in the third row, second column must be less than the value in the third row, third column.

5 Initial Values (Pre-filled Cells)

This section defines some initial values that are already set in certain cells of the grid.

```
formulas(initial_values).
  f(0,0) = 1.
  f(1,1) = 2.
  f(3,3) = 3.
end_of_list.
```

- `f(0,0) = 1`: The value in the top-left cell is fixed as 1.
- `f(1,1) = 2`: The value in the second row, second column is fixed as 2.
- `f(3,3) = 3`: The value in the bottom-right corner is fixed as 3.

These initial values provide starting points for the puzzle and reduce the solution space for the solver.

6 Distinctness Constraints

The distinctness constraints ensure that the numbers in each row and column are unique.

```
list(distinct).
  [f(0,0), f(0,1), f(0,2), f(0,3)].
  [f(1,0), f(1,1), f(1,2), f(1,3)].
  [f(2,0), f(2,1), f(2,2), f(2,3)].
  [f(3,0), f(3,1), f(3,2), f(3,3)].
  [f(0,0), f(1,0), f(2,0), f(3,0)].
  [f(0,1), f(1,1), f(2,1), f(3,1)].
  [f(0,2), f(1,2), f(2,2), f(3,2)].
  [f(0,3), f(1,3), f(2,3), f(3,3)].
end_of_list.
```

- The list of distinctness constraints ensures that the values in each row and each column are distinct. For example, the first row must have unique values: `[f(0,0), f(0,1), f(0,2), f(0,3)]`.
- Similarly, each column is required to have unique values, such as `[f(0,0), f(1,0), f(2,0), f(3,0)]`.

7 Conclusion

The MACE4 code described in this document defines a 4x4 puzzle that combines the constraints of a Latin Square and Futoshiki puzzle. The solver ensures that:

- Each number appears exactly once in every row and column.
- Certain inequalities between adjacent cells are satisfied.
- Pre-filled cells are respected.
- All numbers in each row and column are distinct.

This combination of constraints makes the puzzle both challenging and interesting to solve.

Bibliography

[1] A. Groza. *MACE4 and Logic Puzzles*, Technical University of Cluj-Napoca. Available at: `https://users.utcluj.ro/~agroza/puzzles/maloga/codes.html`.

Intelligent Systems Group