Chess-Num - PLOG 2020

FEUP-PLOG, Class 3MIEIC03, Group 3

Faculty of the engineering of the university of Porto

Abstract. This paper is a brief analysis of our solution of the Chess-Num problem, developed in the context of the PLOG U.C. The solution is implemented in sicstus prolog using its finite domain constraints library (clpfd). TODO copiar conclusao para aqui (conclusões e principais resultados)

1 Introduction

In this paper we describe our solution to the Chess-Num problem, which can solve any instance of the puzzle, generate a random solution, and present the result in a human readable way. We start by describing the problem, afterwards we explain our implementation, and then we analyze the solution/approach.

We weren't able to find any other approaches/references to this problem.

2 Problem Description

The Chess-Num problem is a chess-related puzzle in which, given a set of numbered cells in the chess board, one tries to place the six different chess pieces (rook, queen, king, bishop, knight pawn) in such a way that the number of each given cell corresponds to the number of pieces attacking that cell. The source of this problem has a description of this problem and examples of boards and their solution.

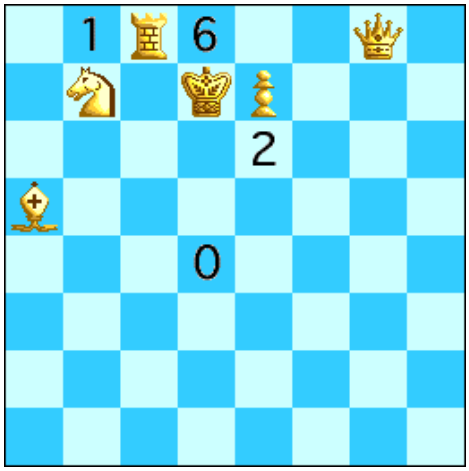

Fig. 1. An example puzzle with four numbered squares and its unique solution.

3 Approach

3.1 Decision Variables

The decision variables correspond to the coordinate pair of each piece. They are all within the domain [0, 7] (inclusive). Furthermore, all of these coordinate pairs are distinct both between each other, as well from the given numbered cells' coordinates.

3.2 Constraints

4 Solution Presentation

The main (outermost) predicates that allow for a problem/solution visualization are the `display_board/1` and the `display_board/2` predicates.

4.1 The `display_board(+NumberedSquares)` predicate

This predicate will draw a chess board with the given numbered squares coordinates showing the given values. This is used to show a problem without its solution. It should be noted that the predicates used to visually represent a solution do so "on-the-fly". This means that only the input data structures are used instead of a *game board* structure being generated and displayed.

The call display_board([[1, 0]-1, [3, 0]-6, [4, 2]-2, [3, 4]-0]). yields the following:

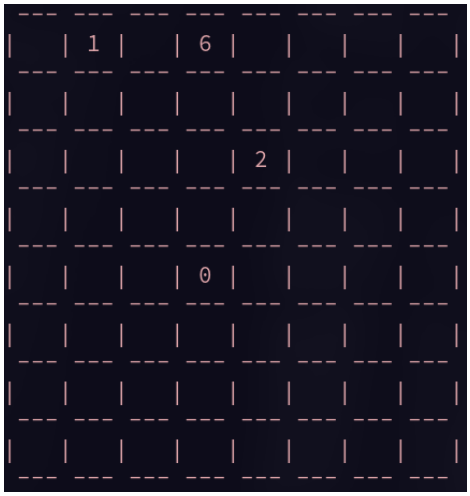

Fig. 2. The textual representation of the puzzle show in fig1 1 (without its solution).

4.2 The `display_board(+NumberedSquares, +Coords)` predicate

Similarly to display_board/1, this predicate will draw a chess board with the given numbered cells. Along side those, the pieces in the given coordinates will also be represented. The representation of each piece is as follows: King - **K**, Queen - **Q**, Rook - **R**, Bishop - **B**, Knight - **Kn**, and Pawn - **P**.

The call display_board([[1, 0]-1, [3, 0]-6, [4, 2]-2, [3, 4]-0], [[3, 1], [6, 0], [2, 0], [0, 3], [1, 1], [4, 1]]). yields the following:

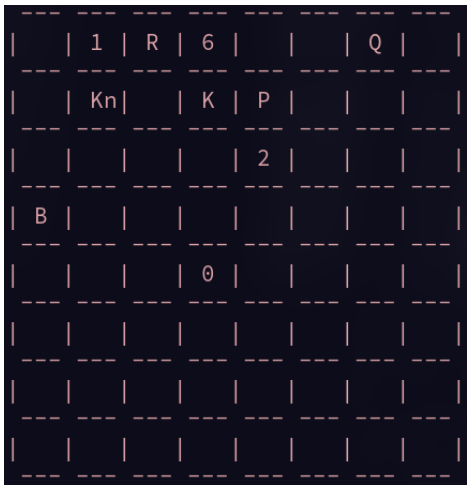

Fig. 3. The textual representation of the puzzle show in fig1 1 (along side its solution).

4.3 Innermost display predicates

// TODO ?

5 Experiments and Results

6 Conclusions and Future Work

References

7 Annex