Completion of Stable Patterns in the Game of Life

Javier Larrosa

LSI-UPC

1 Description of the Problem

The game of *life* is played over an $n \times n$ checkerboard, where each square is called a *cell*. The *neighbors* of a cell are those cells that share one or two corners with it. Note that internal cells have eight neighbors, corner cells have three neighbors, non-corner boundary cells have five neighbors. In the game, the only player places checkers on some cells. If there is a checker on it, the cell is *alive*, else it is *dead*. The interest of the game is to observe how the board evolves iteratively according to the following three rules: (1) if a cell has exactly two living neighbors then its state remains the same in the next iteration, (2) if a cell has exactly three living neighbors then it is alive in the next iteration and (3) if a cell has fewer than two or more than three living neighbors, then it is dead in the next iteration. Although defined in terms of extremely simple rules, the game of life has proven mathematically rich and it has attracted the interest of both mathematicians and computer scientists.

In this project we are interested in stable patterns. A *stable pattern* is a board configuration that remains stable along interations. Considering the rules of the game, it is easy to see that each cell (i, j) must satisfy the following three conditions: (1) if the cell is alive, it must have exactly two or three living neighbors, (2) if the cell is dead, it must not have three living neighbors, and (3) if the cell is at the grid boundary (i.e, $i = 1$ or $i = n$ or $j = 1$ or $j = n$), it cannot be part of a sequence of three consecutive living cells along the boundary. The last condition is motivated because three consecutive living cells at a boundary would produce living cells in hypothetical cells outside the grid. Figure 1 shows two stable patterns.

In this project we consider a (non necessarily stable) $n \times n$ board where some checkers have been placed. We are interested in finding out if we can obtain a stable pattern by adding zero o more new

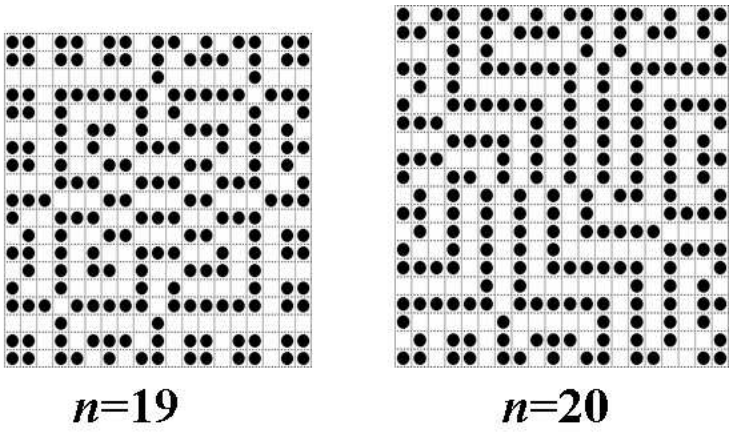

Fig. 1. Two stable patterns.

checkers. For instance, Figure 2 shows on the left a non-stable board and, on the right, a stable completion by adding two checkers

```
0 0 0 0        0 0 0 0
0 0 1 0        0 1 1 0
0 1 0 0        0 1 1 0
0 0 0 0        0 0 0 0
```

Fig. 2. Completion of stable pattern.

The purpose of this project is to model and solve the problem with a SAT solver (such as MiniSat) and a constraint programming solver (such as Comet).

2 Input and Output Format

Problem instances will be given as text files containing only integer values separated by blank spaces. Each line ends with a carry return.

The first line of each file contains two values: the size of the grid n and the number of already placed checkers m. Each one of the following m lines indicates the location of each one of the checkers. Locations are given as coordenates. The top-left corner of the board has coordenate $(1, 1)$ and the bottom-right has coordenate (n, n)

As an example, consider the following input file,

```
4 2
2 3
3 2
```

which represents the board on the left of Figure 2. Solved instances must be recorded in so-called output files having n lines with n numbers each. If the completion is possible, the output will be made of 0's and 1's representing one completion. If the completion is not possible, the output will only contain 0's. As an example, the following output file,

```
0 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0
```

is a solution for the previous input file.

3 What do you have to do:

The first part of the project is concerned with SAT. You have to produce the following pieces of work:

- A SAT encoding for the problem described in a document called "model-description"
- A translator ("BoardToBool") from the previously described input format to dimacs format, consistent with your model.
- A translator ("boolToBoard") from minisat2 solution format to the output format previously described.

A typical sequence of use of the different programs would be:

```
> BoardToBool < problem.txt > boolInstance.cnf
> minisat2 boolInstance.cnf model.txt
> boolToBoard < model.txt > problemSolution.txt
```

The second part of the project is concerned with Constraint Programming. Instructions will be given about what do you have to do.