Chris Gonzalez
Alex Taylor
Theory of Computation
Fall 2013

Solving Minesweeper using Nondeterministic

Introduction:

Minesweeper is a game where players try to solve an $n \times n$ board with x number of mines on the playing field where 0 < x < n^2. The player is able to click on the cells that have not been revealed. If the cell contains a bomb then the player loses and must restart. If there is no bomb in the revealed cell then a number appears. The number represents the amount of mines that are adjacent (left, right, up, down, and diagonally) to the cell that was revealed. Richard Kaye published a proof that the problem of solving any board of size $n \times n$ becomes a NP-complete problem. Meredith Kadlac has since shown that with a Minesweeper board of size $1 \times n$ is solvable in polynomial time. For this project we will be creating an algorithm that replicates the NFA and DFA described in the research paper titled "$2 \times n$ Minesweeper Consistency Problem is in P" by Hu and Lin.

Problem Statement:

For this project we will implement a functional Minesweeper board. The board will take user inputs and when activated, will process the board through a simulated finite automa. The goal of the automa is to prove that a board is consistent and to show the

different consistent boards to the user. The user then may be able to reveal cells on the board and then again process the board through the automa and again show the possible consistent boards.

Implementation:

We found source code for a minesweeper program coded in Python so we decided to use it to implement our solution. The program outputs a 9 x 9 minesweeper board to the terminal with all cells marked unknown (?). It then prompts the user to enter the coordinates (for example column a row 5 would be entered as “a5”) of the first cell and builds a game board based on that input with that cell revealed as well as any neighboring cells containing a 0. To flag a cell you add “f” after the cell coordinates (so if cell d3 is a mine you would enter “d3f” to set a flag on it). The user can then continue revealing cells or enter “nfa” to cut out a 2 x n section of the board to input into the NFA to attempt to solve. The NFA will then output any consistent boards. If there are none, the user can input a different 2 x n section or continue playing the game normally. If there is exactly one consistent board then the user can update the game board to reflect the consistent board. If there are multiple consistent boards, the user must look at all the boards and determine if there are any cells that are exactly the same across all consistent boards. If there are, the user can update the game board with the cell(s) that were the same across all consistent boards. This process is repeated continuously until the user wins the game by flagging all the mines and revealing all cells that are not mines

Supported Features:

- A playable randomized minesweeper board.
- Internal NFA that will process select minesweeper 2 x n boards.

NFA:

Input alphabet: {0?, 1?, 2?, 3?, 4?, 5?, ?B, ??, 00, 11, 1B, 22, 2B, 33, 3B, 44, 4B, 5B, BB}

The input alphabet consists of groupings of two cells along the columns.

A full transition table can be found in the PDF included in our project.

States can be interpreted using the table included in the PDF.

1	2	2

Figure 1.1

The input tape for Figure 1 would be {1?, 2?, 2?}. If the board were to be flipped upside down the outcome would be the same but for this specific NFA the input must be reversed to match the input alphabet.

When the NFA processes the input it produces a derivation tree. The tree describes

the state it came from, the input it received, and processes two groups of two characters that describe the input. For example if the board in Figure 1 is processed we would get the following two derivation trees.

1. 0 , 1? -> 25 (11?B)

   25 , 2? -> 16 (21?0)

   16 , 2? -> 19 (21?B)

2. 0 , 1? -> 14 (10?0)

   14 , 2? -> 19 (21?B)

   19 , 2? -> 27 (22?B)

Each state consists of two groups of two delimiters. In number 1 above the delimiter '11?B' can be read '11', '?B' which means that the first cell must be a 1 and the second covered cell must be a bomb. Two consistent boards that are produced are listed in Figure 1.2 and Figure 1.3

B		B
1	2	2

Figure 1.2

	B	B
1	2	2

Figure 1.2

In the above boards we cannot immediately determine where the bombs should be. However the last cell in the first row is a bomb between both consistent boards. We can reliably mark that a bomb and move on with the board.

Note: an input such as ‘32’ in not possible see Figure 2. No matter how you place three bombs the input ‘2’ will be over fulfilled.

	3	
	2	

Figure 2.

Since the original board is greater than 2 x n we need to do some math with the board surrounding our 2 x n board. Take for example the board in Figure 3.

2	2	2

1	F	1

Figure 3.

If we were to take only the top two rows and feed them into the NFA we would get an incorrect answer. Since there is a flag below the second row we can immediately deduct 1 from the value of all the twos. In essence the input tape would consists of {?1,?1,?1}.

Note: We were not able to introduce an algorithm that fixes the input tape when given an incorrect input. Take Figure 4 for example.

		2
1	1	

Figure 4.

For this board we run into the problem of an incorrect input. The input tape would be {?1, ?1, 2?}. '2?' is not a valid input for the NFA. If we attempt to flip each input to correct '2?' to '?2' which is a valid input the other two input would be '1?' which is not a valid input. What we can deduce from this board is that there must be a flag outside the given 2 x n board that also touches the 2. The input tape must be adjusted based on this assumption.