International Institute of Information Technology
Bangalore

Modelling Assignment

2016-CS 703

Author 1:
Aneesha Devulapally
IMT2013013

Author 2:
Rishabh Manoj
IMT2013035

November 18, 2016

Contents

List of Figures

1 Project problem

The problem we worked on was Loyd's puzzle, a puzzle that challenges its players to slide the tiles in a certain (initial) configuration to achieve a predefined (final) configuration. The initial and the final configurations are shown below.

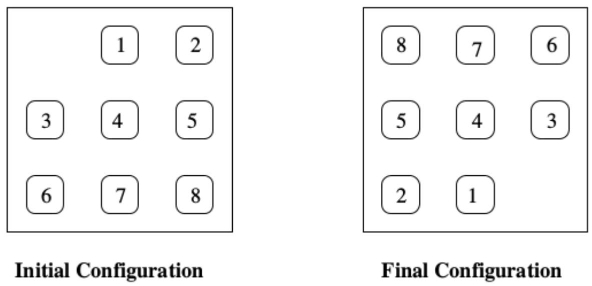


Figure 1: Initial and Final Configuration of the Puzzle

2 Modelling the problem

Loyd's puzzle has an NxK grid with $N.K - 1$ numbered tiles and a blank tile. There are two arrays, h and v, that keep track of the horizontal and vertical positions of the tiles respectively, such that the position of tile i is given by $h[i]$ and $v[i]$. Tile 0 represents the blank tile. Position $h[i] = 1$ and $v[i] = 1$ is the lowest left corner of the puzzle. Here, we define the action *move* [actions *up, down, left* and *right*] with reference to Tile 0. For example, if Tile 1 is to the left of Tile 0, the move *left* results in Tile 0 moving *left* and Tile 1 moving *right*. The initialization of all the tiles is done according to the given initial configuration. The next position of each tile is depended on *move* of Tile 0.

Next position of Tile 0 is defined as follows.

```
next(h[0]) := case
        (move = u) & !(h[0] = N) : h[0] + 1;
        (move = d) & !(h[0] = 1) : h[0] - 1;
                TRUE : h[0];
        esac;

next(v[0]) := case
        (move = l) & !(v[0] = 1) : v[0] - 1;
        (move = r) & !(v[0] = K) : v[0] + 1;
                TRUE : v[0];
        esac;
```

a. If the action *move* is *up* and Tile *0* is not in the last row, then move *up* ($h[0] + 1$)

b. If the action *move* is *down* and Tile *0* is not in the first row, then move *down* ($h[0] - 1$)

c. If the action *move* is *left* and Tile *0* is not in the first column, then move *left* ($v[0] - 1$)

d. If the action *move* is *right* and Tile *0* is not in the last column, then move *right* ($v[0] + 1$)

e. Otherwise, remain in the same position.

Next position of Tile i (1-8) is defined as follows.

```
next(h[i]) := case
    (move = d) & !(h[0] = 1) & (v[0]=v[i]) & (h[i] = h[0]-1) |
    (move = u) & !(h[0] = N) & (v[0]=v[i]) & (h[i] = h[0]+1)
         : h[0];
    TRUE : h[i];
    esac;

next(v[i]) := case
    (move = r) & !(v[0] = K) & (h[0]=h[i]) & (v[i] = v[0]+1) |
    (move = l) & !(v[0] = 1) & (h[0]=h[i]) & (v[i] = v[0]-1)
         : v[0];
    TRUE : v[i];
    esac;
```

(Note: The action *move* is with reference to Tile 0)

a. If the action *move* is *down*, Tile *0* is not in the first row, the vertical positions of Tile *0* and Tile i are equal and Tile i is below Tile *0* ($h[i] = h[0] - 1$), **OR**, if the action *move* is *down*, Tile *0* is not in the last row, the vertical positions of Tile *0* and Tile i are equal and Tile i is above Tile *0* ($h[i] = h[0] + 1$), then horizontal position of Tile i is equal to that of Tile *0*.

b. If the action *move* is *right*, Tile *0* is not in the last column, the horizontal positions of Tile *0* and Tile i are equal and Tile i is to the right of Tile *0* ($v[i] = v[0] + 1$), **OR**, if the action *move* is *left*, Tile *0* is not in the first column, the horizontal positions of Tile *0* and Tile i are equal and Tile i is to the left of Tile *0* ($v[i] = v[0] - 1$), then horizontal position of Tile i is equal to that of Tile *0*.

c. Otherwise, remain in the same position.

3 Goal and CTL specification

We have defined our *goal* as the conjunction of positions of all the tiles according to the final configuration, and the CTL formula as **!EF(*goal*)**, which is "Always, globally the defined *goal* is false". When the code is run, it will verify whether the specification is True or False. If the specification is False, it will output a counter example to prove it.

4 Results

The code runs showing the number of BDD nodes while going through all the (10^7) possible states and declares that the specification is false, as shown in Fig:2.

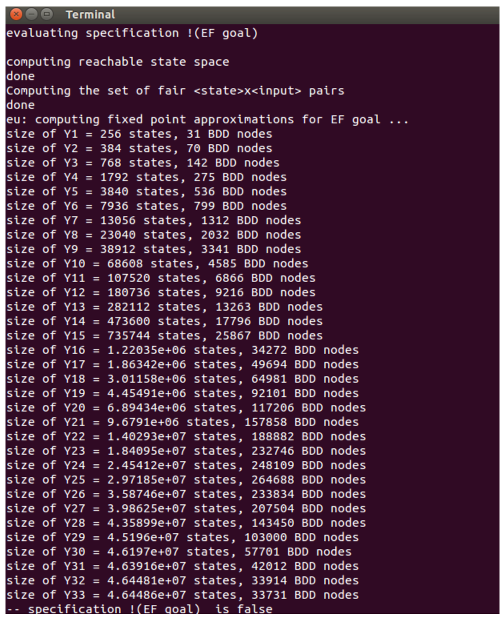
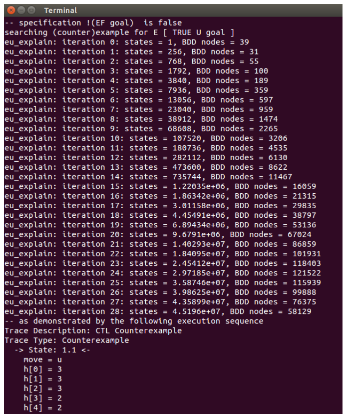
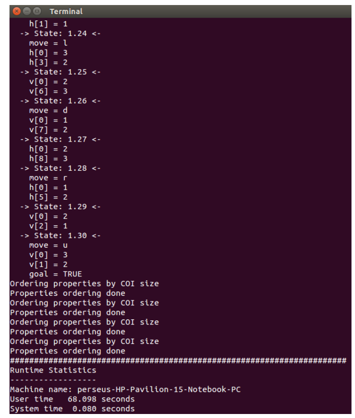


Figure 2: Computation of **fixed point** approximation for CTL specification

It also gives a path where at the end the defined goal is true (Fig:3a, Fig:3b).



(a) Counter Example



(b) Execution Sequence

Figure 3: CTL Counter Example and Execution Sequence

5 Conclusion

For a 3x3 Loyd's puzzle, we can write a program to model check the problem using a brute force method but for any higher dimension, the number of BDD nodes exponentially increases which makes the brute force method an infeasible option. In such cases, we have to delve into the field of *Artificial Intelligence* which makes use of heuristic techniques as this problem is **NP-complete**.