Formal Verification and Synthesis

Final Project

Ziv Chaba 326681285

Hila Cohen 326298445

https://github.com/SokobanFormalVerification.git

Introduction

Sokoban, Japanese for "warehouse keeper", is a transport puzzle created by Hiroyuki Imabayashi in 1980. The goal of the game is simple, the warehouse keeper must push the boxes to designated locations in the warehouse.

Part 1

First, based on the Sokoban's rules, we will define a FDS for a general $n \times m$ Sokoban board, using the XSB format.

$$D = (V, \theta, \rho, J, C)$$

When, in our case-

$$V = \{board: n \times m \ \in \mathbb{N} \times \mathbb{N}, whLoc_x \ \in \mathbb{N}, whLoc_y \ \in \mathbb{N}, turn: \{start, l, u, r, d\}\}$$

When $whLoc$ stands for the warehouse keeper's location on the board.

$$\theta = \{turn = start\}$$

And the rest will be determined for every new run.

Assume $i, j \in \mathbb{N}$, the indexes of the board 2D array are $0, \dots, n-1$ and $0, \dots, m-1$ as usual, and as discussed on class, on the board's edges there are always walls (#), so the warehouse keeper or any box cannot be there.

$$\rho_{board} =$$

$$\begin{aligned}(((1 \le j \le n-3) \wedge (1 \le i \\ \le n-2) \wedge (whLoc_x = j) \wedge \left(whLoc_y = i\right) \wedge (board[i][j+1] = -) \\ \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \left(whLoc_y{}' = i\right) \\ \wedge (board[i][j+1]' = @))\end{aligned}$$

$$\Big((1 \le j \le n-3) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j+1] = .) \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j+1]' = +)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-4) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j+1] = *) \wedge (board[i][j+2] = -) \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j+1]' = +) \wedge (board[i][j+2]' = \$)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-4) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j+1] = *) \wedge (board[i][j+2] = .) \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j+1]' = +) \wedge (board[i][j+2]' = *)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-4) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j+1] = \$) \wedge (board[i][j+2] = -) \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j+1]' = @) \wedge (board[i][j+2]' = \$)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-4) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j+1] = \$) \wedge (board[i][j+2] = .) \wedge (turn' = r) \wedge (whLoc_x' = j+1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j+1]' = @) \wedge (board[i][j+2]' = *)\Big)$$

$$\vee$$

$$\Big((2 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j-1] = -) \wedge (turn' = l) \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j-1]' = @)\Big)$$

$$\vee$$

$$\Big((2 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \wedge (board[i][j-1] = .) \wedge (turn' = l) \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j-1]' = +)\Big)$$

$$\vee$$

$$\Big((3 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \\ \wedge (board[i][j-1] = *\,) \wedge (board[i][j-2] = -) \wedge (turn' = l) \\ \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j-1]' = +) \\ \wedge (board[i][j-2]' = \$)\Big)$$

$$\vee$$

$$\Big((3 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \\ \wedge (board[i][j-1] = *\,) \wedge (board[i][j-2] = .) \wedge (turn' = l) \\ \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j-1]' = +) \\ \wedge (board[i][j-2]' = *)\Big)$$

$$\vee$$

$$\Big((3 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \\ \wedge (board[i][j-1] = \$\,) \wedge (board[i][j-2] = -) \wedge (turn' = l) \\ \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y' = i\big) \wedge (board[i][j-1]' = @) \\ \wedge (board[i][j-2]' = \$)\Big)$$

$$\vee$$

$$\Big((3 \le j \le n-2) \wedge (1 \le i \le n-2) \wedge (whLoc_x = j) \wedge \big(whLoc_y = i\big) \\ \wedge (board[i][j-1] = \$\,) \wedge (board[i][j-2] = .) \wedge (turn' = l) \\ \wedge (whLoc_x' = j-1) \wedge \big(whLoc_y{}' = i\big) \wedge (board[i][j-1]' = @) \\ \wedge (board[i][j-2]' = *)\Big)$$

$$\vee$$

$$\Big((1 \le i \le n-2) \wedge (1 \le j \le n-3) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \\ \wedge (board[i][j+1] = -) \wedge (turn' = d) \wedge (whLoc_x' = i) \\ \wedge \big(whLoc_y{}' = j+1\big) \wedge (board[i][j+1]' = @)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (1 \le i \le n-3) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \\ \wedge (board[i+1][j] = .) \wedge (turn' = d) \wedge (whLoc_x' = i+1) \\ \wedge \big(whLoc_y{}' = j\big) \wedge (board[i+1][j]' = +)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (1 \le i \le n-4) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \\ \wedge (board[i+1][j] = *\,) \wedge (board[i+2][j] = -) \wedge (turn' = d) \\ \wedge (whLoc_x' = i+1) \wedge \big(whLoc_y' = j\big) \wedge (board[i+1][j]' = +) \\ \wedge (board[i+2][j]' = \$)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (1 \le i \le n-4) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i+1][j] = *\,) \wedge (board[i+1][j] = .\,) \wedge (turn' = d) \wedge (whLoc_x' = i+1) \wedge \big(whLoc_y' = j\big) \wedge (board[i+1][j]' = +) \wedge (board[i+2][j]' = *)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (1 \le i \le n-4) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i+1][j] = \$\,) \wedge (board[i+2][j] = -) \wedge (turn' = d) \wedge (whLoc_x' = i+1) \wedge \big(whLoc_y' = j\big) \wedge (board[i+1][j]' = @) \wedge (board[i+2][j]' = \$)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (1 \le i \le n-4) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i+1][j] = \$\,) \wedge (board[i+2][j] = .\,) \wedge (turn' = d) \wedge (whLoc_x' = i+1) \wedge \big(whLoc_y{}' = j\big) \wedge (board[i+1][j]' = @) \wedge (board[i+2][j]' = *)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (2 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i-1][j] = -) \wedge (turn' = u) \wedge (whLoc_x' = i-1) \wedge \big(whLoc_y' = j\big) \wedge (board[i-1][j]' = @)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (2 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i-1][j] = .\,) \wedge (turn' = u) \wedge (whLoc_x' = i-1) \wedge \big(whLoc_y' = j\big) \wedge (board[i-1][j]' = +)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (3 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i-1][j] = *\,) \wedge (board[i-2][j] = -) \wedge (turn' = u) \wedge (whLoc_x' = i-1) \wedge \big(whLoc_y' = j\big) \wedge (board[i-1][j]' = +) \wedge (board[i-2][j]' = \$)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (3 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \wedge (board[i-1][j] = *\,) \wedge (board[i-1][j] = .\,) \wedge (turn' = u) \wedge (whLoc_x' = i-1) \wedge \big(whLoc_y' = j\big) \wedge (board[i]-1[j]' = +) \wedge (board[i-2][j]' = *)\Big)$$

$$\vee$$

$$\Big((1 \le j \le n-2) \wedge (3 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \\ \wedge (board[i-1][j] = \$\,) \wedge (board[i-2][j] = -) \wedge (turn' = u) \\ \wedge (whLoc'_x = i-1) \wedge \big(whLoc'_y = j\big) \wedge (board[i-1][j]' = @) \\ \wedge (board[i-2][j]' = \$)\Big)$$

$$\vee$$

$$((1 \le j \le n-2) \wedge (3 \le i \le n-2) \wedge (whLoc_x = i) \wedge \big(whLoc_y = j\big) \\ \wedge (board[i-1][j] = \$\,) \wedge (board[i-2][j] = .\,) \wedge (turn' = u) \\ \wedge (whLoc'_x = i-1) \wedge \big(whLoc'_y = j\big) \wedge (board[i-1][j]' = @) \\ \wedge (board[i-2][j]' = *)))$$

When each line describes a different case:

1. **Right** movement to a place with floor (-).
2. Right movement to a place with a goal (.).
3. Right movement to a place with a box on a goal (*), when to its right there is floor.
4. Right movement to a place with a box on a goal (*), when to its right there is a goal.
5. Right movement to a place with a box ($), when to its right there is floor.
6. Right movement to a place with a box ($), when to its right there is a goal.
7. **Left** movement to a place with floor (-).
8. Left movement to a place with a goal (.).
9. Left movement to a place with a box on a goal (*), when to its left there is floor.
10. Left movement to a place with a box on a goal (*), when to its right there is a goal.
11. Left movement to a place with a box ($), when to its right there is floor.
12. Left movement to a place with a box ($), when to its right there is a goal.
13. **Down** movement to a place with floor (-).
14. Down movement to a place with a goal (.).
15. Down movement to a place with a box on a goal (*), when to its right there is floor.
16. Down movement to a place with a box on a goal (*), when to its right there is a goal.
17. Down movement to a place with a box ($), when to its right there is floor.
18. Down movement to a place with a box ($), when to its right there is a goal.
19. **Up** movement to a place with floor (-).
20. Up movement to a place with a goal (.).
21. Up movement to a place with a box on a goal (*), when to its right there is floor.
22. Up movement to a place with a box on a goal (*), when to its right there is a goal.
23. Up movement to a place with a box ($), when to its right there is floor.
24. Up movement to a place with a box ($), when to its right there is a goal.

Moreover, the Justice J is that boxes ($) are always not in a corner or edge walls and other deadlock situations (for example - at least in two sides of the boxes there are walls (#) or a wall (#) and a box ($) which near a blocking wall to the same side as the box).

The compassion (C) is that infinite times of right turns yields infinite times of left turns, and vice versa. In addition, infinite times of up turns yields infinite times of down turns, and vice versa.

The general temporal logic specification for a win of the Sokoban board is that finally, there are no places with box ($) on the board (as discussed in class, the number of boxes on a board always equals to the number of goals).

$$F\ (board[0][0] \neq .\$\ \wedge \ldots \wedge\ board[n-1][m-1] \neq\ \$)$$

Part 2

Now, in continuation to the answers of Part 1, we will use Python in order to automate definition of given input boards into SMV models. These models contain both the model and the temporal logic formulae defining a win. Link to the GitHub repository.

For that, we created two scripts for both building the SMV files for the given boards using automation and running these SMV files - BuildNuXmv.py RunNuXmv.py respectively.

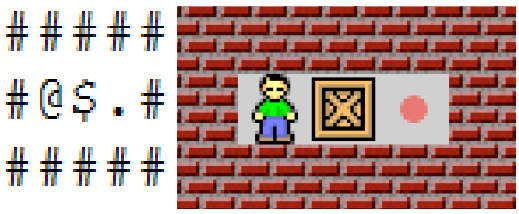
In this part, we used the given XSB boards. We first write the above scripts, and then created in the work directory a new directory – rootDir, in which all the Sokoban XSB inputs and all the related output (generated SMV codes, outputs of those, and also a results' summery for them for each part). In the work directory we also located the nuXmv.exe program, by which we will check our specifications.

Specifically, for this part, we created the Part2XSBDir, Part2SMVDir, Part2Outputs directories.

We turned the given boards into Sokoban XSB files, and then put them inside the Part 2 input directory, Part2XSBDir. We ran the RunNuXmv.py, which created us the reflected-automated SMV boards, and it also put them all inside Part2SMVDir directory. Then, the script also iterates over all the SMV boards and run each one of them, using the command [nuxmv.exe filePath], and the generated output files are located inside Part2Outputs directory.
Moreover, in order to make this program as much easy-to-use and automated, the script also automatically generates a summary file, inside Part2Outputs directory, of all the runs, and indicates for each board if it is solvable (winnable), and if so, the LURD format steps in order to win.

The results:

Board 1:

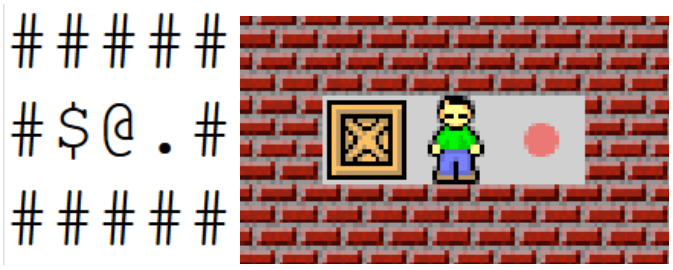



```
-- specification !( F win)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    move = start
    lOption = FALSE
    uOption = FALSE
    rOption = TRUE
    dOption = FALSE
    whLocX = 1
    whLocY = 1
    numCurrentGoals = 1
    board[0][0] = wall
    board[0][1] = wall
    board[0][2] = wall
    board[0][3] = wall
    board[0][4] = wall
    board[1][0] = wall
    board[1][1] = whKeeper
    board[1][2] = box
    board[1][3] = goal
    board[1][4] = wall
    board[2][0] = wall
    board[2][1] = wall
    board[2][2] = wall
    board[2][3] = wall
    board[2][4] = wall
    win = FALSE
    m = 5
    n = 3
  -> State: 1.2 <-
    move = r
    whLocX = 2
    numCurrentGoals = 0
    board[1][1] = floorSign
    board[1][2] = whKeeper
    board[1][3] = boxOnGoal
    win = TRUE
```

```
-- Loop starts here
-> State: 1.3 <-
  move = l
  lOption = TRUE
  rOption = FALSE
  whLocX = 1
  board[1][1] = whKeeper
  board[1][2] = floorSign
-> State: 1.4 <-
  move = r
  lOption = FALSE
  rOption = TRUE
  whLocX = 2
  board[1][1] = floorSign
  board[1][2] = whKeeper
-> State: 1.5 <-
  move = l
  lOption = TRUE
  rOption = FALSE
  whLocX = 1
  board[1][1] = whKeeper
  board[1][2] = floorSign
```

The winning solution (LURD Format): r.

Board 2:

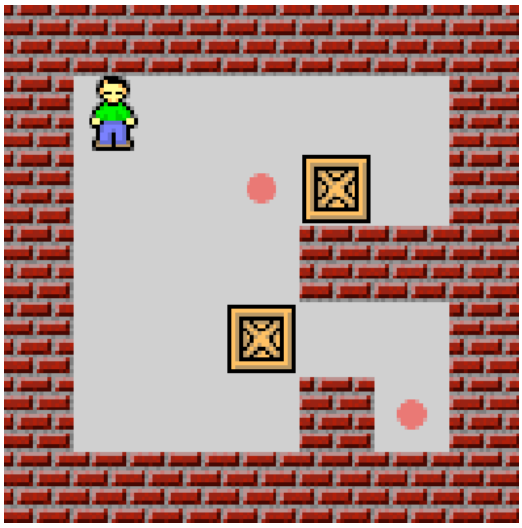



```
-- specification !( F win)  is true
```

The board is not winnable.

Board 3:

```
#######
#@----#
#--.$-#
#---###
#--$--#
#---#.#
#######
```

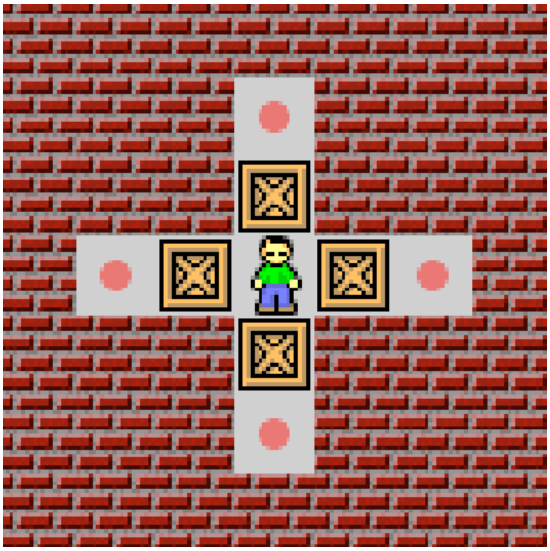


-- specification !(F win) is true

The board is not winnable.

Board 4:

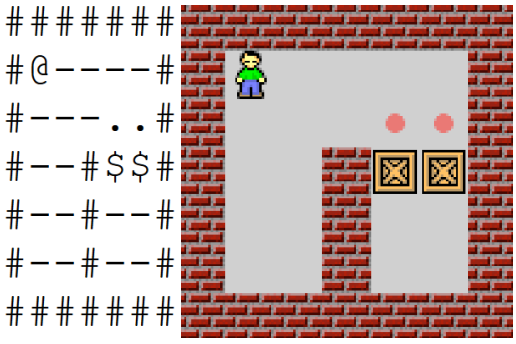
```
#######
###.###
###$###
#.$@$.#
###$###
###.###
#######
```



```
-- specification !( F win)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    move = start
    lOption = TRUE
    uOption = TRUE
    rOption = TRUE
    dOption = TRUE
    whLocX = 3
    whLocY = 3
    numCurrentGoals = 4
    board[0][0] = wall
    board[0][1] = wall
    board[0][2] = wall
    board[0][3] = wall
    board[0][4] = wall
    board[0][5] = wall
    board[0][6] = wall
    board[1][0] = wall
    board[1][1] = wall
    board[1][2] = wall
    board[1][3] = goal
    board[1][4] = wall
    board[1][5] = wall
    board[1][6] = wall
    board[2][0] = wall
    board[2][1] = wall
    board[2][2] = wall
    board[2][3] = box
    board[2][4] = wall
    board[2][5] = wall
    board[2][6] = wall
    board[3][0] = wall
    board[3][1] = goal
    board[3][2] = box
    board[3][3] = whKeeper
    board[3][4] = box
    board[3][5] = goal
    board[3][6] = wall
    board[4][0] = wall
    board[4][1] = wall
    board[4][2] = wall
    board[4][3] = box
    board[4][4] = wall
    board[4][5] = wall
    board[4][6] = wall
    board[5][0] = wall
    board[5][1] = wall
    board[5][2] = wall
    board[5][3] = goal
    board[5][4] = wall
    board[5][5] = wall
    board[5][6] = wall
    board[6][0] = wall
    board[6][1] = wall
    board[6][2] = wall
    board[6][3] = wall
    board[6][4] = wall
    board[6][5] = wall
    board[6][6] = wall
    win = FALSE
    m = 7
    n = 7
  -> State: 1.2 <-
    move = u
    whLocY = 2
    numCurrentGoals = 0
    board[1][3] = boxOnGoal
    board[2][3] = whKeeper
    board[3][3] = floorSign
  -> State: 1.3 <-
    move = d
    lOption = FALSE
    uOption = FALSE
    rOption = FALSE
    whLocY = 3
    board[2][3] = floorSign
    board[3][3] = whKeeper
  -> State: 1.4 <-
    lOption = TRUE
    uOption = TRUE
    rOption = TRUE
    whLocY = 4
    board[3][3] = floorSign
    board[4][3] = whKeeper
    board[5][3] = boxOnGoal
  -> State: 1.5 <-
    move = u
    lOption = FALSE
    rOption = FALSE
    dOption = FALSE
    whLocY = 3
    board[3][3] = whKeeper
    board[4][3] = floorSign
  -> State: 1.6 <-
    move = l
    lOption = TRUE
    rOption = TRUE
    dOption = TRUE
    whLocX = 2
    board[3][1] = boxOnGoal
    board[3][2] = whKeeper
    board[3][3] = floorSign
  -> State: 1.7 <-
    move = r
    lOption = FALSE
    uOption = FALSE
    dOption = FALSE
    whLocX = 3
    board[3][2] = floorSign
    board[3][3] = whKeeper
  -> State: 1.8 <-
    lOption = TRUE
    uOption = TRUE
    dOption = TRUE
    whLocX = 4
    board[3][3] = floorSign
    board[3][4] = whKeeper
    board[3][5] = boxOnGoal
    win = TRUE
  -- Loop starts here
  -> State: 1.9 <-
    move = l
    uOption = FALSE
    rOption = FALSE
    dOption = FALSE
    whLocX = 3
    board[3][3] = whKeeper
    board[3][4] = floorSign
  -> State: 1.10 <-
    move = r
    uOption = TRUE
    rOption = TRUE
    dOption = TRUE
    whLocX = 4
    board[3][3] = floorSign
    board[3][4] = whKeeper
  -> State: 1.11 <-
    move = l
    uOption = FALSE
    rOption = FALSE
    dOption = FALSE
    whLocX = 3
    board[3][3] = whKeeper
    board[3][4] = floorSign
```

The winning solution (LURD Format): uddulrr.

Board 5:

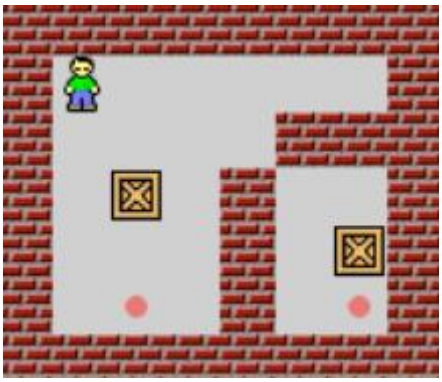



```
-- specification !( F win)  is true
```

The board is not winnable.

Board 6:

```
########
#@-----#
#----###
#-$-#--#
#---#-$#
#-.-#-.#
########
```

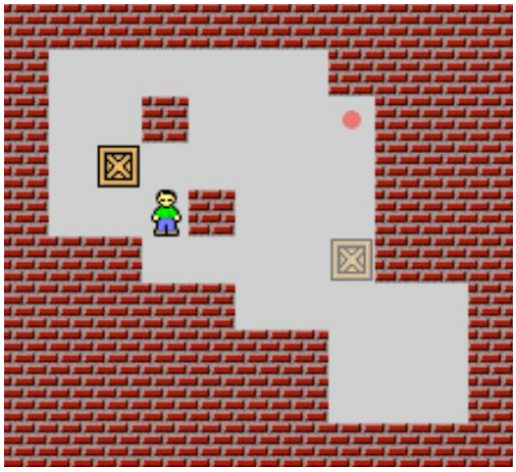


-- specification !(F win) is true

The board is not winnable.

Board 7:

```
##########
#------####
#--#---.###
#-$-----###
#--@#---###
###----*###
#####-----#
#######--##
#######---#
##########
```



```
-- specification !( F win)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    move = start
    lOption = TRUE
    uOption = TRUE
    rOption = FALSE
    dOption = TRUE
    whLocX = 3
    whLocY = 4
    numCurrentGoals = 1
    board[0][0] = wall
    board[0][1] = wall
    board[0][2] = wall
    board[0][3] = wall
    board[0][4] = wall
    board[0][5] = wall
    board[0][6] = wall
    board[0][7] = wall
    board[0][8] = wall
    board[0][9] = wall
    board[0][10] = wall
    board[1][0] = wall
    board[1][1] = floorSign
    board[1][2] = floorSign
    board[1][3] = floorSign
    board[1][4] = floorSign
    board[1][5] = floorSign
    board[1][6] = floorSign
    board[1][7] = wall
    board[1][8] = wall
    board[1][9] = wall
    board[1][10] = wall
    board[2][0] = wall
    board[2][1] = floorSign
    board[2][2] = floorSign
    board[2][3] = wall
    board[2][4] = floorSign
    board[2][5] = floorSign
    board[2][6] = floorSign
    board[2][7] = goal
    board[2][8] = wall
    board[2][9] = wall
    board[2][10] = wall
    board[3][0] = wall
    board[3][1] = floorSign
    board[3][2] = box
    board[3][3] = floorSign
    board[3][4] = floorSign
    board[3][5] = floorSign
    board[3][6] = floorSign
    board[3][7] = floorSign
    board[3][8] = wall
    board[3][9] = wall
    board[3][10] = wall
    board[4][0] = wall
    board[4][1] = floorSign
    board[4][2] = floorSign
    board[4][3] = whKeeper
    board[4][4] = wall
    board[4][5] = floorSign
    board[4][6] = floorSign
    board[4][7] = floorSign
    board[4][8] = wall
    board[4][9] = wall
    board[4][10] = wall
    board[5][0] = wall
    board[5][1] = wall
    board[5][2] = wall
    board[5][3] = floorSign
    board[5][4] = floorSign
    board[5][5] = floorSign
    board[5][6] = floorSign
    board[5][7] = boxOnGoal
    board[5][8] = wall
    board[5][9] = wall
    board[5][10] = wall
    board[6][0] = wall
    board[6][1] = wall
    board[6][2] = wall
    board[6][3] = wall
    board[6][4] = wall
    board[6][5] = floorSign
    board[6][6] = floorSign
    board[6][7] = floorSign
    board[6][8] = floorSign
    board[6][9] = floorSign
    board[6][10] = wall
    board[7][0] = wall
    board[7][1] = wall
    board[7][2] = wall
    board[7][3] = wall
    board[7][4] = wall
    board[7][5] = wall
    board[7][6] = wall
    board[7][7] = floorSign
    board[7][8] = floorSign
    board[7][9] = floorSign
    board[7][10] = wall
    board[8][0] = wall
    board[8][1] = wall
    board[8][2] = wall
    board[8][3] = wall
    board[8][4] = wall
    board[8][5] = wall
    board[8][6] = wall
    board[8][7] = floorSign
    board[8][8] = floorSign
    board[8][9] = floorSign
    board[8][10] = wall
    board[9][0] = wall
    board[9][1] = wall
    board[9][2] = wall
    board[9][3] = wall
    board[9][4] = wall
    board[9][5] = wall
    board[9][6] = wall
    board[9][7] = wall
    board[9][8] = wall
    board[9][9] = wall
    board[9][10] = wall
    win = FALSE
    m = 11
    n = 10
  -> State: 1.2 <-
    move = l
    whLocX = 2
    numCurrentGoals = 0
    board[4][2] = whKeeper
    board[4][3] = floorSign
  -> State: 1.3 <-
    rOption = TRUE
    dOption = FALSE
    whLocX = 1
    board[4][1] = whKeeper
    board[4][2] = floorSign
  -> State: 1.4 <-
    move = u
    lOption = FALSE
    whLocY = 3
    board[3][1] = whKeeper
    board[4][1] = floorSign
  -> State: 1.5 <-
    move = r
    dOption = TRUE
    whLocX = 2
    board[3][1] = floorSign
    board[3][2] = whKeeper
    board[3][3] = box
  -> State: 1.6 <-
    lOption = TRUE
    whLocX = 3
    board[3][2] = floorSign
    board[3][3] = whKeeper
    board[3][4] = box
  -> State: 1.7 <-
    uOption = FALSE
    whLocX = 4
    board[3][3] = floorSign
    board[3][4] = whKeeper
    board[3][5] = box
  -> State: 1.8 <-
    uOption = TRUE
    dOption = FALSE
    whLocX = 5
    board[3][4] = floorSign
    board[3][5] = whKeeper
    board[3][6] = box
  -> State: 1.9 <-
    dOption = TRUE
    whLocX = 6
    board[3][5] = floorSign
    board[3][6] = whKeeper
    board[3][7] = box
  -> State: 1.10 <-
    move = d
    rOption = FALSE
    whLocY = 4
    board[3][6] = floorSign
    board[4][6] = whKeeper
  -> State: 1.11 <-
    move = r
    rOption = TRUE
    whLocX = 7
    board[4][6] = floorSign
    board[4][7] = whKeeper
  -> State: 1.12 <-
    move = u
    rOption = FALSE
    whLocY = 3
    board[2][7] = boxOnGoal
    board[3][7] = whKeeper
    board[4][7] = floorSign
    win = TRUE
  -- Loop starts here
  -> State: 1.13 <-
    move = l
    uOption = FALSE
    whLocX = 6
    board[3][6] = whKeeper
    board[3][7] = floorSign
  -> State: 1.14 <-
    move = r
    uOption = TRUE
    rOption = TRUE
    whLocX = 7
    board[3][6] = floorSign
    board[3][7] = whKeeper
  -> State: 1.15 <-
    move = l
    uOption = FALSE
    rOption = FALSE
    whLocX = 6
    board[3][6] = whKeeper
    board[3][7] = floorSign
```

The winning solution (LURD Format): llurrrrrdru.

Part 3

In this part, we would like to measure the performance of both nuXmv's and SAT Solver engines.

We created inside the rootDir directory, the directories part3XSBDir, part3SMVDir, Part3Output similarly to the previous part and we located four solvable (winnable) XSB Sokoban boards inside part3XSBDir- boards number 1, 4, 7 which we saw in part 2, and also a larger more complex Sokoban board, board number 8:

```
##########
#--$-.-####
#--#---.###
##-$----###
#--@#---###
###----$###
#####-----#
#######---#
#######.--#
##########
```

We ran the two solver engines (both in interactive mode, for BDD we used the commands go and check_ltlspec, and for SAT we used the commands go_bmc and check_ltlspec_bmc -k 100, 100 can be replaced by the automation engineer for any other bound, can be determined by requirements) on each one of those boards, and got the following results (can be seen in details in summaryOutput.txt file which inside the Part3Output directory):

Board	BDD [sec]	SAT [sec]	LURD Winning Steps
Board 1	0.123	0.089	r
Board 4	1.143	0.959	uddulrr
Board 7	239.687	10.748	llurrrrrdru
Board 8	444.927	88.231	luuurrlldrrrrdrudddd

We can see that we can separate the results into two groups- Boards 1,4 and Boards 7,8.

For Boards 1,4 the runtimes of BDD and SAT are about to be the same. In this specific run, the SAT was faster, but in most of the runs, actually the BDD was a little bit faster.

However, for Boards 7,8 the runtime of the SAT engine was remarkably lower (and so-faster), than the BDD engine, and the memory usage was also less.

These results are reasonable. We know that BDD seems to be preferred than SAT in cases of small specification models, as Board 1 and Board 2. However, the SAT is preferred both in runtime and memory usage than BDD in cases of large specification models, such as complex boards as Board 7 and Board 8.

Part 4

Now, we will break the problem into sub-problems by solving the boards iteratively. We choose to solve in each iteration for at least one more box than the previous iteration. The temporal logic formulae used for each iteration is ! $F(numCurrentGoals < numOriginalGoals)$ where $numOriginalGoals$ is the number of boxes ($) on the board at the first of each iteration, and $numCurrentGoals$ is the current-updated state of number of remaining boxes on the board of each iteration.

Every iteration, we force to reduce the number of remaining boxes on the board which we get from the previous iteration. We are also protected from unwanted cases of false-negative (which can be caused by a box that was in iteration I was solved, in iteration i+1 is not solved and also placed in deadlock places like corners and edges walls for example) by our transition function.

We created inside the rootDir directory, the directories part4XSBDir, part4SMVDir, Part4Output similarly to the previous part and we located four solvable (winnable) XSB Sokoban boards inside part4XSBDir- boards number 1, 4, 7 and 8.

We ran for each iteration of each board the SAT engine (since we saw it is preferred on large and complex boards, especially 7 and 8), similarly to the previous part, and got the following results (can be seen in details in summaryOutput.txt file which inside the Part4Output directory):

Board	Number of Iterations	Runtime per Iteration [sec]	Total Runtime [sec]
Board 1	1	[0.114]	0.114
Board 4	4	[0.265, 0.272, 0.259, 0.298]	1.095
Board 7	1	[9.257]	9.257
Board 8	3	[2.614, 4.164, 47.350]	54.129

First, we can see that the number of the iterations for each board is equal to its number of boxes, which is reasonable and fit our specifications.

Moreover, we can see that on board 1, 4, 7 we got similar total runtimes as Part 3. This phenomenon is not surprising for board 1 and board 7, since we have only one box in them. In board 4, we see that in non-complex and small boards, like board 4, the iterative solution's runtime is similar to the non-iterative one.

Regarding to board 8, which is similar to board 7, a large and complex board, we see that with the iterative solution in this part, we significantly improving our runtime performance – almost doubling the speed. We can conclude that for large and complex boards, the iterative solution is more efficient and achieve a better performance than the non-iterative solution.

Conclusions

In this project we solved the Sokoban game by Formal Verification and Synthesis methods.

On Part 1, we defined a generic Sokoban game, and used it to build a complete automation for the next parts.

On Part 3, we compared between of nuXmv's BDD and SAT Solver engines, and found out that on larger-complex specifications, the SAT engine can achieve a better performance than the BDD engine.

On Part 4, we suggested an iterative algorithm for solving the Sokoban game, and found out that especially on large and complex Sokoban boards, the iterative method achieves a better performance than the non-iterative method which we saw in the previous parts.