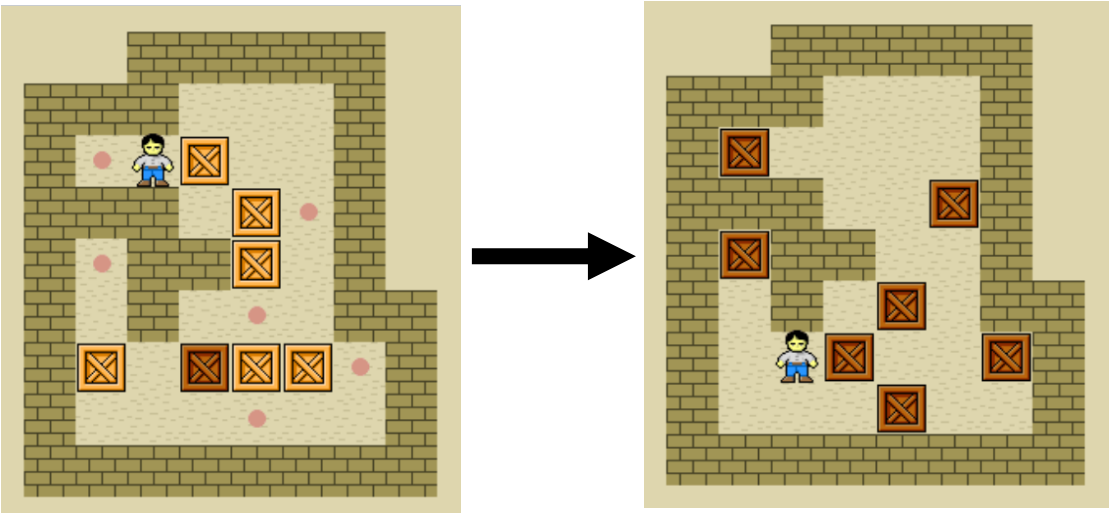
Formal Verification Final Project

Yaniv Hajaj

Maya Naor



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.

The warehouse is depicted as a grid with walls creating a labyrinth. The following are rules that must be respected:

- Warehouse keeper can only move horizontally or vertically in the grid, one cell at a time.
- Boxes can only be pushed, not pulled, into an empty space.
- Warehouse keeper and boxes cannot enter "wall" cells.

- Include a link to your GitHub repo in your PDF submission.

https://github.com/YanivHajaj/Formal-Verification-Final-Project/tree/main

Part 1

1. Define an FDS for a general $n \times m$ Sokoban board. Use XSB format to describe the board.

הלוח מתקבל בפורמט XSB לדוגמה:

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

ערך התחלתי של כל מיקום בלוח: נקבע מפורמט הקלט XSB על ידי מציאת הסמל (לדוגמה @ או +).
הלוח מיוצג על ידי מערך דו ממדי board שבתוכו שני מספרים שלמים,
[עמודה][שורה] מייצגים את המיקום של כל אובייקט.
כל מספר שלם יכול לקבל ערכים מ-0 עד n-1 עבור שורות ו-0 עד m-1 עבור עמודות,
כאשר n ו-m הם מספר השורות והעמודות בלוח בהתאמה.

move מייצג את כיוון התנועה ('L', 'R', 'U', 'D')

הוספנו זוג סדור (DY,DX) = (תזוזה בציר אופקי, תזוזה בציר אנכי) לתוך סט התזוזות האפשריות movments שמתאים את המיקום החדש שאותו נבדוק ביחס למיקום הנוכחי.

```
transitions = '\t'
movements = {
    'l': (0, -1),
    'r': (0, 1),
    'u': (-1, 0),
    'd': (1, 0)
}
```

למשל אם נבחר ב-R אז יש התייחסות לבדיקה של המקום הבא בשורה (DX=1).

נגדיר FDS:

משתנים V:

board[i][j] מייצג את סוג האלמנט בלוח Sokoban במיקום (i, j) ויכול לקבל כל אלמנט מתוך הסט:

Symbol	Definition
@	warehouse keeper
+	warehouse keeper on goal
$	box
*	box on goal
#	wall
.	goal
-	floor

move יכול לקבל את הערכים של כיוון התנועה 'L', 'R', 'U', 'D' וממנו נגזר סט התזוזות האפשריות movements. נשים לב שבעזרת movements והcurrent_move נדע להוסיף\להחסיר את ההיסט הדרוש כדי לבדוק את המיקום המתאים של התאים הבאים board[i ± DX][j ± DY].

מעברים δ:

עבור כל עמדת לוח (i, j) נגדיר את המצב הבא בהתבסס על המצב הנוכחי וכיוון התנועה.

המהלכים האפשריים עבור כל אריח בלוח (מקרה ותוצאה תואמת):

0. קיר (#): <u>נשאר זהה למה שהיה ללא קשר למהלך</u>:

```
# the current cell is a wall - can't change
if board[i][j] == '#':
    transitions += f'next(board[{i}][{j}]) := Wall;\n\t\t'
else:
    transitions += f'\n\t\tnext(board[{i}][{j}]) := \n\t\tcase\n'
```

1. warehouse keeper (@) או warehouse keeper on goal (+), התא הבא בכיוון התנועה הוא קיר (#):

התא נשאר זהה למה שהיה.

```
for currentMove, (dx, dy) in movements.items():
    1# the current cell is the WarehouseKeeper or KeeperOnGoal and the next cell is wall
    transitions += (
        f'\t\t\t\t(board[{i}][{j}] = WarehouseKeeper | board[{i}][{j}] = KeeperOnGoal) & move = {currentMove} & board[{i + dx}][{j + dy}] = Wall: board[{i}][{j}];\n')
```

2. warehouse keeper (@), התא הבא בכיוון התנועה הוא רצפה (-) או מטרה (.):

התא הופך לרצפה (-).

```
2# the current cell is WarehouseKeeper and the next cell is floor or goal
transitions += (
    f'\t\t\tboard[{i}][{j}] = WarehouseKeeper & move = {currentMove} & (board[{i + dx}][{j + dy}] = Floor | board[{i + dx}][{j + dy}] = Goal) : Floor;\n')
```

3. warehouse keeper on goal (+), התא הבא בכיוון התנועה הוא רצפה (-) או מטרה (.):

התא הופך למטרה (.).

```
3# the current cell is KeeperOnGoal and the next cell is floor or goal
transitions += (
    f'\t\t\tboard[{i}][{j}] = KeeperOnGoal & move = {currentMove} & (board[{i + dx}][{j + dy}] = Floor | board[{i + dx}][{j + dy}] = Goal) : Goal;\n')
```

4.התא הנוכחי הוא מטרה, warehouse keeper (@) או warehouse keeper on goal (+), נע למטרה (.):

התא הופך ל warehouse keeper on goal (+).

```
4# the current cell is Goal and the WarehouseKeeper or KeeperOnGoal go this cell
transitions += (
    f'\t\t\tboard[{i}][{j}] = Goal & move = {currentMove} & (board[{i - dx}][{j - dy}] = WarehouseKeeper | board[{i - dx}][{j - dy}] = KeeperOnGoal)  : KeeperOnGoal;\n')
```

5.התא הנוכחי הוא ריצפה, warehouse keeper (@) או warehouse keeper on goal (+), נע לריצפה (-):

התא הופך ל warehouse keeper (@).

```
5# the current cell is Floor and the WarehouseKeeper or KeeperOnGoa go this cell
transitions += (
    f'\t\t\tboard[{i}][{j}] = Floor & move = {currentMove} & (board[{i - dx}][{j - dy}] = WarehouseKeeper | board[{i - dx}][{j - dy}] = KeeperOnGoal) : WarehouseKeeper;\n')
```

6. warehouse keeper (@) או warehouse keeper on goal (+), התא הבא הוא תיבה ($) או תיבה on goal (*) והתא הבא(2X) הוא קיר (#) או תיבה($) או תיבה on goal (*):

התא נשאר זהה למה שהיה.

```
if 0 <= i + 2 * dx < len(board) and 0 <= j + 2 * dy < len(board[0]):
    # the current cell is the WarehouseKeeper or KeeperOnGoal, the next cell is box or boxOnGoal and the next cell is wall, box or boxOnGoal
    transitions += (
        f'\t\t\t\t(board[{i}][{j}] = WarehouseKeeper | board[{i}][{j}] = KeeperOnGoal) & move = {currentMove} & (board[{i + dx}][{j + dy}] = Box | board[{i + dx}][{j + dy}] = BoxOnGoal) & (board[{i + 2 * dx}][{j + 2 * dy}] = Box | board[{i + 2 * dx}][{j + 2 * dy}] = Wall | board[{i + 2 * dx}][{j + 2 * dy}] = BoxOnGoal) : board[{i}][{j}];\n')
```

7. warehouse keeper (@), התא הבא הוא תיבה ($) או תיבה on goal (*) והתא הבא(2X) הוא ריצפה (-) או מטרה(.) :

התא הופך לריצפה (-).

```
7# the current cell is the WarehouseKeeper, the next cell is box or boxOnGoal and the next cell is Floor or Goal
transitions += (
    f'\t\t\tboard[{i}][{j}] = WarehouseKeeper & move = {currentMove} & (board[{i + dx}][{j + dy}] = Box | board[{i + dx}][{j + dy}] = BoxOnGoal) & (board[{i + 2 * dx}][{j + 2 * dy}] = Floor | board[{i + 2 * dx}][{j + 2 * dy}] = Goal) : Floor;\n')
```

8. on goal warehouse keeper (+), התא הבא הוא תיבה ($) או תיבה on goal (*) והתא הבא(2X) הוא ריצפה (-) או מטרה(.) :

התא הופך למטרה (.).

```
8# the current cell is the KeeperOnGoal, the next cell is box or boxOnGoal and the next cell is Floor or Goal
transitions += (
    f'\t\t\tboard[{i}][{j}] = KeeperOnGoal & move = {currentMove} & (board[{i + dx}][{j + dy}] = Box | board[{i + dx}][{j + dy}] = BoxOnGoal) & (board[{i + 2 * dx}][{j + 2 * dy}] = Floor | board[{i + 2 * dx}][{j + 2 * dy}] = Goal) : Goal;\n')
```

9. תיבה ($) או תיבה on goal (*), התא הבא הוא תיבה ($) או תיבה on goal (*) או קיר (#), והתא הקודם היה warehouse keeper (@) או warehouse keeper on goal (+):

התא נשאר זהה למה שהיה.

```
9# the previous cell is the WarehouseKeeper or KeeperOnGoal, the current cell is box or boxOnGoal and the next cell is wall, box or boxOnGoal
transitions += (
    f'\t\t\tboard[{i}][{j}] = Box | board[{i}][{j}] = BoxOnGoal) & move = {currentMove} & (board[{i + dx}][{j + dy}] = Box | board[{i + dx}][{j + dy}] = BoxOnGoal | board[{i + dx}][{j + dy}] = Wall) & (board[{i - dx}][{j - dy}] = WarehouseKeeper | board[{i - dx}][{j - dy}] = KeeperOnGoal) : board[{i}][{j}];\n')
```

10. תיבה ($), התא הבא הוא רצפה (-) או מטרה (.), והתא הקודם היה warehouse keeper (@) או warehouse keeper on goal (+):

התא הופך ל warehouse keeper (@).

```
10# the current cell is box, the next cell is floor or goal, and the previous cell is WarehouseKeeper or KeeperOnGoal
transitions += (
    f'\t\t\tboard[{i}][{j}] = Box & move = {currentMove} & (board[{i + dx}][{j + dy}] = Floor | board[{i + dx}][{j + dy}] = Goal) & (board[{i - dx}][{j - dy}] = WarehouseKeeper | board[{i - dx}][{j - dy}] = KeeperOnGoal) : WarehouseKeeper;\n')
```

11. תיבה on goal (*), התא הבא הוא רצפה (-) או מטרה (.), והתא הקודם היה warehouse keeper (@) או warehouse keeper on goal (+):

התא הופך לwarehouse keeper on goal (+).

```
11# the current cell is box on goal, the next cell is floor or goal, and the previous cell is WarehouseKeeper or KeeperOnGoal
transitions += (
    f'\t\t\tboard[{i}][{j}] = BoxOnGoal & move = {currentMove} & (board[{i + dx}][{j + dy}] = Floor | board[{i + dx}][{j + dy}] = Goal) & (board[{i - dx}][{j - dy}] = WarehouseKeeper | board[{i - dx}][{j - dy}] = KeeperOnGoal) : KeeperOnGoal;\n')
```

12. ריצפה (-), התא הקודם היה תיבה ($) או תיבה on goal (*), והתא שלפניו היה warehouse keeper (@) או warehouse keeper on goal (+):

התא הופך לתיבה ($).

```
if 0 <= i - 2 * dx < len(board) and 0 <= j - 2 * dy < len(board[0]):
    12# the current cell is floor, the previous cell is box or boxOnGoal, and the previous cell is keeper
    transitions += (
        f'\t\t\tboard[{i}][{j}] = Floor & move = {currentMove} & (board[{i - dx}][{j - dy}] = Box | board[{i - dx}][{j - dy}] = BoxOnGoal) & (board[{i - 2 * dx}][{j - 2 * dy}] = WarehouseKeeper | board[{i - 2 * dx}][{j - 2 * dy}] = KeeperOnGoal) : Box;\n')
```

13. ריצפה (-), התא הקודם היה תיבה ($) או תיבה on goal (*), והתא שלפניו היה warehouse keeper (@) או warehouse keeper on goal (+):

התא הופך לתיבה on goal (*).

```
13# the current cell is goal, the previous cell is box or boxOnGoal, and the previous cell is keeper
transitions += (
    f'\t\t\tboard[{i}][{j}] = Goal & move = {currentMove} & (board[{i - dx}][{j - dy}] = Box | board[{i - dx}][{j - dy}] = BoxOnGoal) & (board[{i - 2 * dx}][{j - 2 * dy}] = WarehouseKeeper | board[{i - 2 * dx}][{j - 2 * dy}] = KeeperOnGoal) : BoxOnGoal;\n')
```

כפי שצויין בתרגול חזרה אין צורך בJ=justice, C=compassion או θ (כי זה לוח כללי ולא מתחילים בהכרח ממקום ספציפי מוגדר).

2. Define a general temporal logic specification for a win of the Sokoban board.

האסטרטגיה לניצחון תהיה שבסופו של דבר לכל tile שהוא מטרה (.) או (+) או (*) יהפוך בסופו של דבר להיות קופסא על מטרה (*).

או באופן פורמלי: F(winning_state) כאשר ה winning_state הוא מחרוזת הבנויה מחיבור של כל המקומות בלוח שהיו מטרה ורוצה לבדוק שהם הפכו ל box on goal:

```
# Initialize an empty string to store the goal conditions
goal_conditions = '''
DEFINE
    winning_state := (''' 

# Iterate over the list of goals
for goal in goals:
    # Add the condition for this goal to the string
    goal_conditions += f"(board[{goal[0]}][{goal[1]}] = BoxOnGoal) & "
```

נשים לב שאנחנו נבדוק בקוד את השלילה : LTLSPEC !(F(winning_state)) שכן נרצה שיראה לנו את הדוגמה הנגדית לשלילה ואז בעצם יפתור לנו את המשחק בצעדי LRUD.

Part 2

1. Using Python, or another coding language, and your answers to Part 1, automate definition of given input boards into SMV models. These models should contain both the model and the temporal logic formulae defining a win.

ניתן לראות את הקוד בGITHUB (קישור בעמוד הראשון)

2. Run each of these models in nuXmv. Indicate the commands you used to run nuXmv, as well as screenshots of the nuXmv outputs.

קובץ הNUXMV רץ על שלושת האפשרויות (בעזרת BDD SAT ובאופן רגיל) ומציג פתרונות של כל אחד מהם.
יש לשים לב כי עבור ההרצה של SAT יש להכניס ערך K המהווה את מספר הצעדים המקסימלי שהמודל ינסה לפתרון (אם לא מצא פתרון בK צעדים אזי יעצור)

כדי להריץ את Nuxmv על לוח:
יש צורך להכניס PATH של לוח קלט (בפורמט XSB) במקום המתאים:

```
# Define the path to your file
path = r'C:\Users\yaniv\OneDrive - Bar-Ilan University\Desktop\formal verification\board.txt'
```

הקוד אוטומטית מריץ את שלושת האפשרויות על ידי הפונקציה run_nuxmv
הפונקציה מקבלת את שם הקובץ שעתיד לרוץ, את MODE (BDD/SAT/regular) ו-K

```
def run_nuxmv(model_filename, engine=None, k=None):...
```

כעת יש להכניס את הנתיב לתקייה שדרכה מריצים קבצי Nuxmv במחשב האישי, התכנית מחליפה את עבודתה לתקייה זו כדי להריץ את הקובץ SMV.

```
os.chdir(r'C:\Users\yaniv\OneDrive - Bar-Ilan University\Desktop\nuXmv-2.0.0-win64\bin')
```

לשים לב שיש להחליף את אותו הנתיב בפונקציה createSMVfile

```
def createSMVfile(board):
    # Specifying the path
    path = r'C:\Users\yaniv\OneDrive - Bar-Ilan University\Desktop\nuXmv-2.0.0-win64\bin'
```

עבור כל האופציות קוד הPython מכין את הקובץ SMV כתלות בדרישה:

ריצה ראשונה הרצה רגילה:

```
# run non interactive mode
run_nuxmv("sokoban.smv")
print_solution("sokoban.out", "regular")
print("\n\n")
```

מתבצעת בנייה של תהליך subproces שיריץ את הקובץ

```
else:
    # Run the command with the model filename
    nuxmv_process = subprocess.Popen(base_command + [model_filename], stdin=subprocess.PIPE,
                                     stdout=subprocess.PIPE, universal_newlines=True)
    output_filename = model_filename.split(".")[0] + ".out"

stdout, _ = nuxmv_process.communicate()
```

עבור הרצות נוספות יש צורך במוד אינטרקטיבי לכן קודם נבנה base_command כתלות באם זה BDD או SAT

```
base_command = [".\\nuXmv.exe"]

# Check the engine type
if engine in ["SAT", "BDD"]:
    base_command.extend(["-int", model_filename])
    # Start the nuXmv process
    nuxmv_process = subprocess.Popen(base_command, stdin=subprocess.PIPE, stdout=subprocess.PIPE,
                                     universal_newlines=True)
```

ריצה שנייה MODE BDD

```
# run BDD mode
run_nuxmv("sokoban.smv", "BDD")
print_solution("sokobanBDD.out", "BDD")
print("\n\n")
```

מכניסה באופן אינטרקטיבי קלטים לתוך התכנית Nuxmv בפורמט הנכון לBDD:

```
elif engine == "BDD":
    nuxmv_process.stdin.write("go\n")
    nuxmv_process.stdin.write("check_ltlspec\n")
    output_filename = model_filename.split(".")[0] + "BDD.out"
```

ריצה שלישית MODE SAT (מבקשת שהמשתמש יכניס קלט K)

מכניסה באופן אינטרקטיבי קלטים לתוך התכנית Nuxmv בפורמט הנכון לSAT:

```
# Send commands based on the engine type
if engine == "SAT":
    nuxmv_process.stdin.write("go_bmc\n")
    nuxmv_process.stdin.write(f"check_ltlspec_bmc -k {k}\n")
    output_filename = model_filename.split(".")[0] + "SAT.out"
```

ולבסוף כדי לצאת מהתכנית רושמת באופן אינטרקטיבי

```
# Quit the nuXmv process
nuxmv_process.stdin.write("quit\n")
```

ריצה אחרונה Iterative Solving MODE (מבקשת שהמשתמש יכניס קלט K ומניחה שכל איטריציה לא לוקחת יותר מ20 צעדים מטעמים של זמן ריצה)

```
outputfile, itertime = run_nuxmv("sokoban.smv", "SAT", 20)
```

בגישה האיטרטיבית שמרנו את כל אריחי המטרה במערך. רצנו על כל אריחי המטרה בלולאה ועבור כל איטריציה בלולאה החלפנו את כל אריחי המטרה באריחים רגילים חוץ ממטרה בודדת שאותה רצינו להשיג.
הרצנו על ידי SAT עד K=20 ואז שמרנו את הלוח החדש שהתקבל וביצענו את אותם שלבים כאשר הלוח החדש שעליו נעבוד הוא הלוח מסוף האיטרציה הקודמת:

```
# remove goals from the board
for goal in goals:
    if board[goal[0]][goal[1]] == '+':
        board[goal[0]][goal[1]] = '@'
    elif board[goal[0]][goal[1]] == '.':
        board[goal[0]][goal[1]] = '-'
    elif board[goal[0]][goal[1]] == '*':
        board[goal[0]][goal[1]] = '$'
```

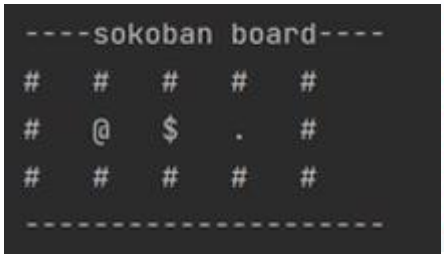
```
index = 1
for goal in goals:
    print(f"iteration number {index} ")
    index = index +1
    # add one goal
    if board[goal[0]][goal[1]] == '@':
        board[goal[0]][goal[1]] = '+'
    elif board[goal[0]][goal[1]] == '-':
        board[goal[0]][goal[1]] = '.'
    elif board[goal[0]][goal[1]] == '$':
        board[goal[0]][goal[1]] = '*'
    print_board(board)
    createSMVfile(board)
    outputfile, itertime = run_nuxmv("sokoban.smv", "SAT", 20)
    totaltime = totaltime + itertime
    currentmoves = print_solution(outputfile, "SAT")
    solution.append(currentmoves)
    # update the board
    with open(outputfile, 'r') as file:
        lines = file.readlines()
```

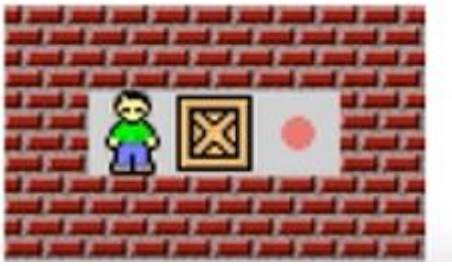
2. Run each of these models in nuXmv. Indicate the commands you used to run nuXmv, as well as screenshots of the nuXmv outputs.

3. For each board, indicate if it is winnable. If it is, indicate your winning solution in LURD format.

בסעיף הקודם ביקשו גם קבצי Nuxmv Outputs לכן עבור כל לוח נראה את הפתרון בLURD. קבצי ה SMV שנוצרו עבור כל לוח מופיעים בתיקייה בGITHUB, לא צירפנו אותם כאן כיוון שהם ארוכים מאוד.

GAMEBOARD-2 (ברור שפתיר)



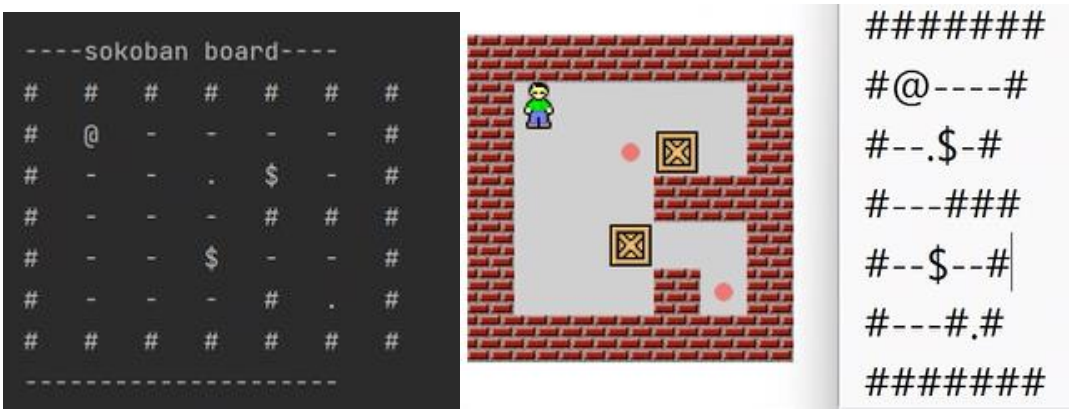



```
the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal)

Output saved to sokoban.out
Performance of the None run: 0.07185125350952148

the solution of this board for the regular run is: ['l', 'r']



Output saved to sokobanBDD.out
Performance of the BDD run: 0.058336734771728516

the solution of this board for the BDD run is: ['l', 'r']



please enter number of steps for SAT run10
Output saved to sokobanSAT.out
Performance of the SAT run: 0.04074430465698242

the solution of this board for the SAT run is: ['r']
```

GAMEBOARD-2 (לא פתיר)


```
main
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal)

Output saved to sokoban.out
Performance of the None run: 0.06660890579223633
there is no solution for the regular run



Output saved to sokobanBDD.out
Performance of the BDD run: 0.051917314529418945
there is no solution for the BDD run



please enter number of steps for SAT run10
Output saved to sokobanSAT.out
Performance of the SAT run: 0.10205507278442383
there is no solution for the SAT run
```

GAMEBOARD-3 (אין פתרון כי לא יכול לדחוף את הקופסא התחתונה למטרה)

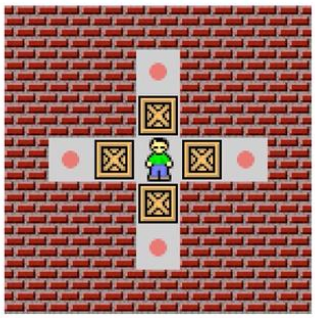


```
Run: main
File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[5][5] = BoxOnGoal)

Output saved to sokoban.out
Performance of the None run: 1.5211827754974365
there is no solution for the regular run



Output saved to sokobanBDD.out
Performance of the BDD run: 1.3619616031646729
there is no solution for the BDD run



please enter number of steps for SAT run10
Output saved to sokobanSAT.out
Performance of the SAT run: 0.7131977081298828
there is no solution for the SAT run
```

GAMEBOARD-3 (פתיר)

```
----sokoban board----
#   #   #   #   #   #   #
#   #   #   .   #   #   #
#   #   #   $   #   #   #
#   .   $   @   $   .   #
#   #   #   $   #   #   #
#   #   #   .   #   #   #
#   #   #   #   #   #   #
--------------------
```

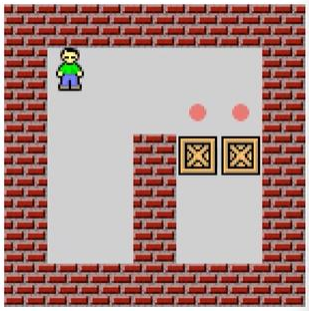
```
#######
###.###
###$###
#.$@$.#
###$###
###.###
#######
```



```
Output saved to sokoban.out
Performance of the None run: 884.5622553825378

the solution of this board for the regular run is: ['l', 'r', 'u', 'd', 'r', 'l', 'd']



Output saved to sokobanBDD.out
Performance of the BDD run: 844.0470674037933

the solution of this board for the BDD run is: ['l', 'r', 'u', 'd', 'r', 'l', 'd']



please enter number of steps for SAT run20
Output saved to sokobanSAT.out
Performance of the SAT run: 0.432020902633667

the solution of this board for the SAT run is: ['d', 'u', 'r', 'l', 'u', 'd', 'l']
```

GAMEBOARD-5 (לא פתיר, אי אפשר להגיע מתחת לקופסאות)

```
----sokoban board----
#   #   #   #   #   #   #
#   @   -   -   -   -   #
#   -   -   -   .   .   #
#   -   -   #   $   $   #
#   -   -   #   -   -   #
#   -   -   #   -   -   #
#   #   #   #   #   #   #
---------------------
```



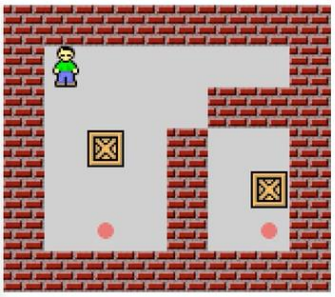
```
#######
#@----#
#---..#
#--#$$#
#--#--#
#--#--#
#######
```

```
the temporal logic formula used is: winning_state := ((board[2][4] = BoxOnGoal) & (board[2][5] = BoxOnGoal)

Output saved to sokoban.out
Performance of the None run: 0.7354826927185059
there is no solution for the regular run



Output saved to sokobanBDD.out
Performance of the BDD run: 1.0947654247283936
there is no solution for the BDD run



please enter number of steps for SAT run14
Output saved to sokobanSAT.out
Performance of the SAT run: 0.9601466655731201
there is no solution for the SAT run
```

GAMEBOARD-6 (לא פתיר, יש חדר שלם שאי אפשר להגיע אליו ולדחוף)

```
----sokoban board----
#   #   #   #   #   #   #
#   @   -   -   -   -   #
#   -   -   -   .   .   #
#   -   -   #   $   $   #
#   -   -   #   -   -   #
#   -   -   #   -   -   #
#   #   #   #   #   #   #
---------------------
```

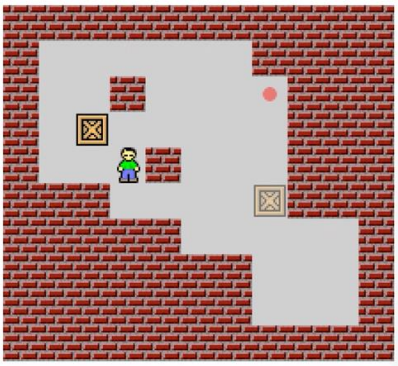
```
########
#@-----#
#----###
#-$-#--#
#---#-$#
#-.-#-.#
########
```



```
the temporal logic formula used is: winning_state := ((board[2][4] = BoxOnGoal) & (board[2][5] = BoxOnGoal)

Output saved to sokoban.out
Performance of the None run: 0.7141892910003662
there is no solution for the regular run



Output saved to sokobanBDD.out
Performance of the BDD run: 0.8431363105773926
there is no solution for the BDD run



please enter number of steps for SAT run10
Output saved to sokobanSAT.out
Performance of the SAT run: 0.9627974033355713
there is no solution for the SAT run
```

GAMEBOARD-7 (פתיר, המפה גדולה לכן הסתכלנו רק על K קטן מ20 כדי שלא יחפש דוגמה לפתרון שהיא מאוד ארוכה וזמן הריצה יהיה ארוך מדי)

```
----sokoban board----
#   #   #   #   #   #   #   #   #   #   #
#   -   -   -   -   -   -   #   #   #   #
#   -   -   #   -   -   -   .   #   #   #
#   -   $   -   -   -   -   -   #   #   #
#   -   -   @   #   -   -   -   #   #   #
#   #   #   -   -   -   -   *   #   #   #
#   #   #   #   #   -   -   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   #   #   #   #
---------------------
```



```
###########
#------####
#--#---,###
#-$-----###
#--@#---###
###----*###
#####-----#
#######---#
#######---#
###########
```

```
File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][7] = BoxOnGoal) & (board[5][7] = BoxOnGoal)

please enter number of steps for SAT run20
Output saved to sokobanSAT.out
Performance of the SAT run: 4.892858028411865

the solution of this board for the SAT run is: ['ל', 'ל', 'ט', 'ר', 'ר', 'ר', 'ר', 'ר', 'd', 'ר', 'ט']
```

Part 3

1. Measure performance of nuXmv's BDD and SAT Solver engines on each of the models.

בצילומי מסך בPART2 ניתן לראות זמני ביצועים בשניות בכל אחד מהלוחות (חוץ מלוח מספר 7 שזמן הריצה שלו היה ארוך מדי בפתרון שהוא לא מוגבל בצעדים)

המדידה של הזמנים נעשתה על ידי דגימת הזמן לפני הרצת קובץ הNUXMV דרך הפייתון ולאחר סיום הרצת הקובץ דגמנו את הזמן שנית וחיסרנו בין הסוף להתחלה:

```
createSMVfile()
# run non interactive mode
timer1 = time.time()
run_nuxmv("sokoban.smv")
timer2 = time.time()
print(f"Performance of the regular run: {timer2 - timer1}")
# run BDD mode
timer1 = time.time()
run_nuxmv("sokoban.smv", "BDD")
timer2 = time.time()
print(f"Performance of the BDD run: {timer2 - timer1}")
# run SAT mode
k = input("please enter number of steps for SAT run")
timer1 = time.time()
run_nuxmv("sokoban.smv", "SAT", k)
timer2 = time.time()
print(f"Performance of the SAT run: {timer2 - timer1}")
```

2. Compare the performance of the two engines. Is one engine more efficient than the other?

חישוב באמצעות SAT יעיל יותר כיוון שהוא בהכרח מוצא את הפתרון הקצר ביותר (בניגוד לריצת BDD שיכולה להוסיף צעדים מיותרים שלא מקדמים את הלוח לפתרון- לדוגמא יכול לעשות כמה צעדים לכיוון קיר שלא מקדמים את השחקן לשום מקום.) בנוסף זמן הריצה שלו קצר יותר בגלל ההגבלה בכמות הצעדים. בהרצות מסויימות (לוח מספר 4 או לוח 7) הוא רץ בשניות בודדות במקום בחצי שעה שלקח לBDD.

Part 4

1. Break the problem into sub-problems by solving the boards iteratively. For example, solve for one box at a time. Indicate the temporal logic formulae used for each iteration.

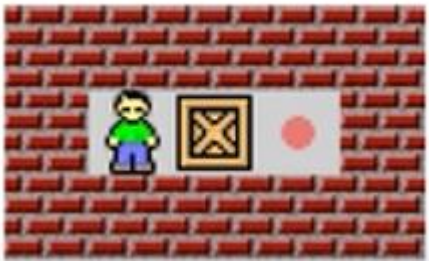
בסעיף זה נבצע גישה איטרטיבית כך שנפתור כל מטרה בודדת אחת אחרי השנייה.
ניתן לראות בכל לוח מה temporal logic של האיטרציה הספציפיצית כמו גם את
זמני הריצה של האיטרציה וזמני ריצה כוללים של סך האיטרציות.

2. Indicate runtime for each iteration, as well as the total number of iterations needed for a given board.

עבור כל לוח ניתן לראות למעלה בתמונה את מספר האיטרציה ולאחר סיום את
מספר האיטרציות הכולל וזמני ריצה כוללים. (בחרנו כמה לוחות לדוגמה)

GAMEBOARD-1

```
----sokoban board----
#   #   #   #   #
#   @   $   .   #
#   #   #   #   #
----------------------
```

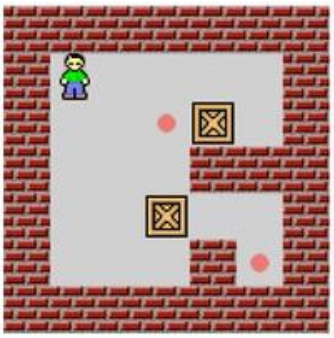


```
#####
#@$.#
#####
```

```
main

**************************** Iteration Mode ****************************
iteration number 1

----sokoban board----
#   #   #   #   #
#   @   $   .   #
#   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.043625593185424805

the solution of this board for the SAT run is: ['r']


the final solution is: [['r']]
total number of iterations is 1
total time for a given board is 0.043625593185424805
```

GAMEBOARD-3 (אין פתרון כי לא יכול לדחוף את הקופסא התחתונה למטרה)

```
----sokoban board----
#   #   #   #   #   #   #
#   @   -   -   -   -   #
#   -   -   .   $   -   #
#   -   -   -   #   #   #
#   -   -   $   -   -   #
#   -   -   -   #   .   #
#   #   #   #   #   #   #
---------------------
```



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

```
main

**************************** Iteration Mode ****************************
iteration number 1

----sokoban board----
#   #   #   #   #   #   #
#   @   -   -   -   -   #
#   -   -   .   $   -   #
#   -   -   -   #   #   #
#   -   -   $   -   -   #
#   -   -   -   #   -   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.413632869720459

the solution of this board for the SAT run is: ['r', 'r', 'r', 'r', 'd', 'l']
iteration number 2
```

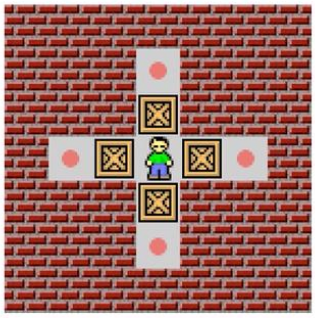
נשים לב שבמקרה זה את המטרה הראשונה אפשר לקיים, ניתן לראות שהBOX העליון ניתן לפתירה.. אך שהתנאי הלוגי מבקש את המטרה התחתונה אין פתרון:

```
iteration number 2

----sokoban board----
#   #   #   #   #   #   #
#   -   -   -   -   -   #
#   -   -   *   @   -   #
#   -   -   -   #   #   #
#   -   -   $   -   -   #
#   -   -   -   #   .   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[5][5] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.777777910232544
there is no solution for the SAT run
```

GAMEBOARD-4 - יש 4 מטרות ולכן נקבל פתרון לאחר 4 איטרציות.

```
----sokoban board----
#   #   #   #   #   #   #
#   #   #   .   #   #   #
#   #   #   $   #   #   #
#   .   $   @   $   .   #
#   #   #   $   #   #   #
#   #   #   .   #   #   #
#   #   #   #   #   #   #
--------------------
```

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



בצילומים ניתן לראות גם את הtemporal logic בכל שלב, כאשר בכל שלב נוסף תא נוסף שמכיל מטרה.

```
**************************** Iteration Mode ****************************
iteration number 1

----sokoban board----
#   #   #   #   #   #   #
#   #   #   .   #   #   #
#   #   #   $   #   #   #
#   -   $   @   $   -   #
#   #   #   $   #   #   #
#   #   #   -   #   #   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.18385791778564453

the solution of this board for the SAT run is: ['u']
```

```
iteration number 2

----sokoban board----
#   #   #   #   #   #   #
#   #   #   *   #   #   #
#   #   #   @   #   #   #
#   .   $   -   $   -   #
#   #   #   $   #   #   #
#   #   #   -   #   #   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal) & (board[3][1] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.2130744457244873

the solution of this board for the SAT run is: ['d', 'ל']
```

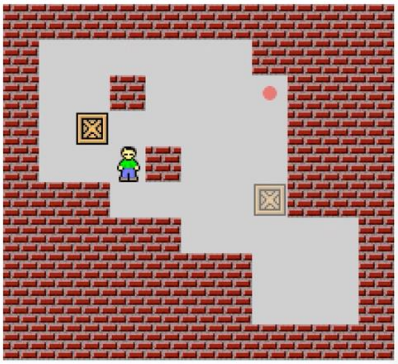
```
iteration number 3

----sokoban board----
#   #   #   #   #   #   #
#   #   #   *   #   #   #
#   #   #   -   #   #   #
#   *   @   -   $   .   #
#   #   #   $   #   #   #
#   #   #   -   #   #   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal) & (board[3][1] = BoxOnGoal) & (board[3][5] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.20763421058654785

the solution of this board for the SAT run is: ['r', 'r']
```

```
iteration number 4

----sokoban board----
#   #   #   #   #   #   #
#   #   #   *   #   #   #
#   #   #   -   #   #   #
#   *   -   -   @   *   #
#   #   #   $   #   #   #
#   #   #   .   #   #   #
#   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[1][3] = BoxOnGoal) & (board[3][1] = BoxOnGoal) & (board[3][5] = BoxOnGoal) & (board[

Output saved to sokobanSAT.out
Performance of the SAT run: 0.2044665813446045

the solution of this board for the SAT run is: ['l', 'd']
```

```
the final solution is: [['u'], ['d', 'l'], ['r', 'r'], ['l', 'd']]
total number of iterations is 4
total time for a given board is 0.8090331554412842
```

GAMEBOARD-7 - יש פתרון לאחר 2 איטרציות

```
----sokoban board----
#   #   #   #   #   #   #   #   #   #   #
#   -   -   -   -   -   -   #   #   #   #
#   -   -   #   -   -   -   .   #   #   #
#   -   $   -   -   -   -   -   #   #   #
#   -   -   @   #   -   -   -   #   #   #
#   #   #   -   -   -   -   *   #   #   #
#   #   #   #   #   -   -   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   #   #   #   #
---------------------
```



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

```
iteration number 1

----sokoban board----
#   #   #   #   #   #   #   #   #   #   #
#   -   -   -   -   -   -   #   #   #   #
#   -   -   #   -   -   -   .   #   #   #
#   -   $   -   -   -   -   -   #   #   #
#   -   -   @   #   -   -   -   #   #   #
#   #   #   -   -   -   -   $   #   #   #
#   #   #   #   #   -   -   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   #   #   #   #
---------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][7] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 1.0374844074249268

the solution of this board for the SAT run is: ['d', 'r', 'r', 'r', 'd', 'r', 'u', 'u', 'u']
```

```
iteration number 2

----sokoban board----
#   #   #   #   #   #   #   #   #   #   #
#   -   -   -   -   -   -   #   #   #   #
#   -   -   #   -   -   -   *   #   #   #
#   -   $   -   -   -   -   @   #   #   #
#   -   -   -   #   -   -   -   #   #   #
#   #   #   -   -   -   -   .   #   #   #
#   #   #   #   #   -   -   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   -   -   -   #
#   #   #   #   #   #   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][7] = BoxOnGoal) & (board[5][7] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 202.16750264167786

the solution of this board for the SAT run is: ['l', 'l', 'l', 'l', 'd', 'l', 'l', 'u', 'r', 'r', 'r', 'r', 'u', 'r', 'd', 'd', 'l', 'd', 'r']
```

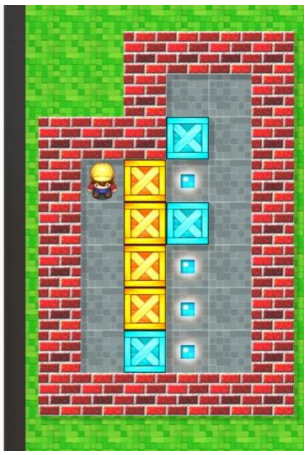
```
the final solution is: [['d', 'r', 'r', 'r', 'd', 'r', 'u', 'u', 'u'], ['l', 'l', 'l', 'l', 'd', 'l', 'l', 'u', 'r', 'r', 'r', 'r', 'u', 'r', 'd',
total number of iterations is 2
total time for a given board is 203.20498704910278
```

קיבלנו פתרון פחות יעיל גם מבחינת מספר צעדים, וגם מבחינת זמן החישוב.

3. Test your iterative solution on larger more complex Sokoban boards.

לקחנו לוח עם 7 מטרות, והרצנו אותו בצורה איטרטיבית וגם באמצעות SAT

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



```
----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   @   $   .   -   #
#   -   $   *   -   #
#   -   $   .   -   #
#   -   $   .   -   #
#   -   *   .   -   #
#   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoal) & (board

please enter number of steps for SAT run20
Output saved to sokobanSAT.out
Performance of the SAT run: 173.01329231262207

the solution of this board for the SAT run is: ['d', 'd', 'd', 'r', 'l', 'd', 'r', 'l', 'u', 'u', 'r', 'l', 'u', 'u', 'r', 'd', 'd', 'd']
```

```
iteration number 1

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   @   $   -   -   #
#   -   $   $   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.13159608840942383

the solution of this board for the SAT run is: []
```

```
iteration number 2

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   @   $   .   -   #
#   -   $   $   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   #   #   #   #   #
---------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.19868206977844238

the solution of this board for the SAT run is: ['r']
```

```
iteration number 3

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   -   @   *   -   #
#   -   $   *   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   #   #   #   #   #
---------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoal)

Output saved to sokobanSAT.out
Performance of the SAT run: 0.12964487075805664

the solution of this board for the SAT run is: []
```

```
iteration number 4

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   -   @   *   -   #
#   -   $   *   -   #
#   -   $   .   -   #
#   -   $   -   -   #
#   -   $   -   -   #
#   #   #   #   #   #
---------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoal) & (board[

Output saved to sokobanSAT.out
Performance of the SAT run: 0.26133227348327637

the solution of this board for the SAT run is: ['l', 'd', 'd', 'r']
```

```
iteration number 5

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   -   -   *   -   #
#   -   $   *   -   #
#   -   @   *   -   #
#   -   $   .   -   #
#   -   $   -   -   #
#   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoal) & (bo

Output saved to sokobanSAT.out
Performance of the SAT run: 0.16967082023620605

the solution of this board for the SAT run is: ['l', 'd', 'r']
```

```
iteration number 6

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   -   -   *   -   #
#   -   $   *   -   #
#   -   -   *   -   #
#   -   @   *   -   #
#   -   *   -   -   #
#   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoal) & (

Output saved to sokobanSAT.out
Performance of the SAT run: 0.15427947044372559

the solution of this board for the SAT run is: []
```

```
iteration number 7

----sokoban board----
#   #   #   #   #   #
#   #   #   -   -   #
#   #   #   *   -   #
#   -   -   *   -   #
#   -   $   *   -   #
#   -   -   *   -   #
#   -   @   *   -   #
#   -   *   .   -   #
#   #   #   #   #   #
--------------------

File sokoban.smv has been created in C:\Users\mayan\OneDrive\שולחן העבודה\nuXmv-2.0.0-win64\bin

the temporal logic formula used is: winning_state := ((board[2][3] = BoxOnGoal) & (board[3][3] = BoxOnGoal) & (board[4][3] = BoxOnGoa

Output saved to sokobanSAT.out
Performance of the SAT run: 4.913246154785156

the solution of this board for the SAT run is: ['l', 'd', 'r', 'l', 'u', 'u', 'u', 'u', 'r', 'd', 'd', 'd']
```

```
the final solution is: [[], ['r'], [], ['l', 'd', 'd', 'r'], ['l', 'd', 'r'], [], ['l', 'd', 'r', 'l', 'u', 'u', 'u', 'u', 'r', 'd', 'd', 'd']]
total number of iterations is 7
total time for a given board is 5.958451747894287
```

ניתן לראות כי בצורה האיטרטיבית (סה"כ 5.99 שניות) קיבלנו פתרון בזמן יותר קצר בצורה משמעותית ביחס לSAT (173.01 שניות), אבל האיטרטיבי (20 צעדים) לא הפתרון האופטימלי מבחינת כמות צעדים ביחס לSAT (18 צעדים בלבד).