פרויקט מסכם
אימות פורמלי וסינתזה

תאריך: 19.05.2024

מגישים :
נועם דיאמנט 208520262
אורה רחל ווצלר 212058689

מבוא

בפרויקט זה, נתבקשנו לכתוב תוכנית פייתון, המייצרת עבור לוח סוקובאן נתון בפורמט XSB, את מודל ה – smv המתאים לו, ומריצה את תוכנת ה –nuXmv על מודל זה, כדי לבדוק האם הלוח פתיר. במידה והלוח פתיר, נתבקשנו להדפיס את התוצאות שיובילו לפתרון הלוח. זאת ועוד, נתבקשנו להשוות בין זמני הריצה של המנועים השונים, ובין זמני הריצה של פתרון איטרטיבי ולא איטרטיבי. התוצאות של החלקים השונים מוצגות להלן.

הגיטהאב של הפרויקט נמצא בכתובת הבאה:

https://github.com/NoamDiamant52/FVS

מיכל והלל, תודה על כל הסמסטר! היה קורס ממש מעניין 😊

חלק ראשון

Part 1

1. Define an FDS for a general $n \times m$ Sokoban board. Use XSB format to describe the board.
2. Define a general temporal logic specification for a win of the Sokoban board.

שאלה 1

נגדיר את ה - FDS של לוח הסוקובאן:

$$D = \{V, \theta, \rho, J, C\}$$

V – משתני המערכת – קיימים שני סוגי משתנים:
משתנים ששייכים ללוח הסוקובאן: $n * m$ משתנים שהם תאים בלוח, כאשר כל תא בלוח מהווה משתנה יחיד. נתאר את הלוח כמערך דו ממדי עם $n * m$ תאים. כל משתנה של תא בלוח יראה כך: $SokobanBoard\ [i][j]$, שהוא התא בשורה ה- i ובעמודה ה – j.
כל משתנה בלוח יכול לקבל את אחד מבין הערכים הבאים[1]:

$$\{@, +, \$, *, \#, ., -\}$$

בנוסף, יש לנו משתנה המתאר את תנועת השומר:

$$direction \in \{u, d, r, l\}$$

נגדיר זאת גם בצורה מתמטית:

$$V = \{SokobanBoard\ [i][j] \in \{@, +, \$, *, \#, ., -\}: \forall 0 \le i \le n, 0 \le j \le m\} \cup direction \in \{u, d, r, l\}$$

θ - מצב התחלתי של המערכת – הלוח הנתון בתחילת כל משחק בפורמט XSB כחלק מהנתונים שניתנים כקלט לתוכנית (ראו הוראות הרצה).

ρ - פונקציית המעברים - המעברים האפשריים של המערכת שבנינו הינם:
פונקציית המעברים של תנועת השומר היא לא דטרמיניסטית, ולכן תהיה:

```
next(direction) := {l, u, r, d};
```

פונקציית המעברים של התאים בלוח:
בחלק זה של פונקציית המעברים, הקצנו פונקציית מעברים נפרדת לכל תא בלוח, כלומר לכל $SokobanBoard\ [i][j]$ ישנה פונקציית מעברים נפרדת משלו.
כאשר נריץ את קוד הפייתון שכתבנו, הוא מייצר את פונקציית המעברים הרלוונטית עבור כל תא בלוח לפי החוקיות שתתואר להלן.
אם הפייתון מזהה כי המצב ההתחלתי בלוח של תא $[i][j]$ כלשהו הוא קיר ('#'), אזי הפייתון יציב לו בתור פונקציית מעבר את rho_i (פונקצייה זו היא פונקציית ה – $idle$ שלא משנה את מצבו של התא בלוח), כיוון שהתא לעולם לא ישתנה:

```
next(SokobanBoard[i][j]) := hashtag;
```

כעת לפני שנמשיך למקרה שבו התא המדובר בלוח אינו קיר, נעיר משהו לשם בהירות ההסבר בהמשך. נשים לב כי כאשר תא מסוים בלוח אינו קיר, הוא יכול להיות או רצפה שאין עליה מטרה, או רצפה שיש עליה מטרה (רק אחת מהאופציות אפשרית, לא שתיהן). בכל אחד משני המצבים הללו, יכול להיות שיש על מקום זה בלוח את השחקן, או את הקופסה,

---

[1] לתשומת לב: מכיוון שחלק מהתווים הם תווים שמורים ב nuXmv, השתמשנו ב nuXmv בקבוצה הבאה: {dollar, asterisk, hashtag, at, plus, dot, dash}. לשם הנוחות, בקובץ הזה נשתמש בתווים עצמם.

או שאין עליו כלום. לכן חילקנו את המצבים הנותרים (אחרי שטיפלנו באופציה של הקיר), לשתי קבוצות:

$$floor\_states = \{-, @, \$\}$$

$$goal\_states = \{., +, *\}$$

בנוסף, על מנת לחסוך זמן ריצה של ה nuXmv, כאשר כתבנו את התוכנית שמייצרת את פונקציית המעברים בפייתון, בדקנו כבר בפייתון אילו מעברים לא יהיו אפשריים לעולם. לדוגמה עבור תא מסויים (שהוא לא קיר) שמימינו יש קיר, התנועה ימינה מהתא הזה תמיד תהיה לא אפשרית, ולכן לא רשמנו בדיקה כזאת ב nuXmv (ישנם עוד מקרים, כמו לדוגמא תא שהשכן מדרגה 2 שלו הוא קיר, אזי מהתא הזה השומר לא יוכל לדחוף קופסא וכו'). המעבר שיתבצע במקרה הזה, כמו בעוד מקרים בהם תא מסויים לא ישתנה (כמו תא שהשומר בכלל לא קרוב אליו, או מקרה של דחיפה של קופסא לכיוון קופסא), יהיה המעבר של rho_i. כלומר באופן ברירת מחדל עבור כל המצבים <u>שלא נכתבו</u> כאפשריים עבור אותו התא, התא ישאר במצבו הנוכחי.

כעת נפרט כיצד כתבנו את החלק עבור המצבים האפשריים בפונקציית המעברים עבור כל תא בלוח: עבור כל תא בלוח, הפייתון יציב לו את פונקציית המעברים הבאה, כאשר תוך כדי הצבה, עבור כל תא שנבדק כאן,
$([i][j], [i+1][j], [i+2][j], [i][j+1] \dots)$, הפייתון מראש יבדוק האם מדובר בתא שהבסיס שלו הוא רצפה ללא מטרה, ולכן בכל תנאי הוא יבדוק ויציב רק אחד מהאפשרויות המתאימות מתוך הקבוצה:

$$floor\_states = \{-, @, \$\}$$

אחרת, אם הבסיס של התא הוא מטרה, הפייתון יזהה זאת ויציב אפשרות מתאימה מתוך הקבוצה:

$$goal\_states = \{., +, *\}$$

להלן מוצגת דוגמה כללית עבור מיקום כללי כלשהו בלוח סוקובאן נתון. יש לשים לב שבקבצים האמייתים מופיעים מספרים במקום i, j כמובן, ובנוסף, כל מעבר מצוין באופן חד חד ערכי (כלומר, לא קיימים בקבצי smv שלנו אופרטור or, אלא מתייחסים לכל מקרה בנפרד לפי עניינו). מעברים אפשריים אמיתי בקובץ smv מוצגים לדוגמה כך:

```
SokobanBoard[3][7] = at & direction = l & SokobanBoard[3][6] = dash : dash;
SokobanBoard[3][4] = dash & direction = r & SokobanBoard[3][3] = plus : at;
```

הדוגמה המוצגת מטה היא רק המחשה לפונקציונליות של המעברים, אך בפועל המעברים מוצגים באופן חד חד ערכי כנ"ל.

```
    next(board[i][j]) :=
        case
            -- case keeper
            SokobanBoard[i][j] = (at | plus) & direction = l & SokobanBoard[i][j - 1] = (dash | dot) :
(dash | dot);
            SokobanBoard[i][j] =  (at | plus) & direction = l & SokobanBoard[i][j-1] = (dollar | asterisk)
& SokobanBoard[i][j-2] = (dash | dot) : (dash | dot);
            SokobanBoard[i][j] =  (at | plus) & direction = r & SokobanBoard[i][j+1] = (dash | dot) : (dash
| dot);
            SokobanBoard[i][j] =  (at | plus) & direction = r & SokobanBoard[i][j+1] = (dollar | asterisk)
& SokobanBoard[i][j+2] = (dash | dot) : (dash | dot);
            SokobanBoard[i][j] =  (at | plus) & direction = d & SokobanBoard[i+1][j] = (dash | dot) : (dash
| dot);
            SokobanBoard[i][j] =  (at | plus) & direction = d & SokobanBoard[i+1][j] = (dollar | asterisk)
& SokobanBoard[i+2][j] = (dash | dot) : (dash | dot);
```

```
            SokobanBoard[i][j] =  (at | plus) & direction = u & SokobanBoard[i-1][j] = (dash | dot) : (dash
| dot);
            SokobanBoard[i][j] =  (at | plus) & direction = u & SokobanBoard[i-1][j] = (dollar | asterisk)
& SokobanBoard[i-2][j] = (dash | dot) : (dash | dot);



            -- case box
            SokobanBoard[i][j] = (dollar | asterisk) & direction = l & SokobanBoard[i][j-1] = (dash | dot)
& SokobanBoard[i][j+1] =  (at | plus):  (at | plus);
            SokobanBoard[i][j] = (dollar | asterisk) & direction = r & SokobanBoard[i][j+1] = (dash | dot)
& SokobanBoard[i][j-1] =  (at | plus):  (at | plus);
            SokobanBoard[i][j] = (dollar | asterisk) & direction = u & SokobanBoard[i-1][j] = (dash | dot)
& SokobanBoard[i+1][j] =  (at | plus):  (at | plus);
            SokobanBoard[i][j] = (dollar | asterisk) & direction = d & SokobanBoard[i+1][j] = (dash | dot)
& SokobanBoard[i+1][j] =  (at | plus):  (at | plus);



            -- case floor
            SokobanBoard[i][j] = (dash | dot) & direction = l & SokobanBoard[i][j+1] =  (at | plus) :  (at
| plus);
            SokobanBoard[i][j] = (dash | dot) & direction = l & SokobanBoard[i][j+1] = (dollar | asterisk)
& SokobanBoard[i][j+2] =  (at | plus) : (dollar | asterisk);
            SokobanBoard[i][j] = (dash | dot) & direction = u & SokobanBoard[i+1][j] =  (at | plus) :  (at
| plus);
            SokobanBoard[i][j] = (dash | dot) & direction = u & SokobanBoard[i+1][j] = (dollar | asterisk)
& SokobanBoard[i+2][j] =  (at | plus) : (dollar | asterisk);
            SokobanBoard[i][j] = (dash | dot) & direction = r & SokobanBoard[i][j-1] =  (at | plus) :  (at
| plus);
            SokobanBoard[i][j] = (dash | dot) & direction = r & SokobanBoard[i][j-1] = (dollar | asterisk)
& SokobanBoard[i][j-2] =  (at | plus) : (dollar | asterisk);
            SokobanBoard[i][j] = (dash | dot) & direction = d & SokobanBoard[i-1][j] =  (at | plus) :  (at
| plus);
            SokobanBoard[i][j] = (dash | dot) & direction = d & SokobanBoard[i-1][j] = (dollar | asterisk)
& SokobanBoard[i-2][j] =  (at | plus) : (dollar | asterisk);



            -- rho_i
            TRUE:

                case
                    SokobanBoard[i][j] = (dash | dot) : (dash | dot);
                    SokobanBoard[i][j] =  (at | plus) :  (at | plus);
                    SokobanBoard[i][j] = (dollar | asterisk) : (dollar | asterisk);
                    -- to avoid nuXmv error. SHOULD NOT HAPPEN!!
                    TRUE : hashtag;

                esac;
```

```
esac;
```

$J = \phi$ – *justice* - המשחק אמור להסתיים לאחר מספר סופי של צעדים, ולכן אין מצב שאמור לקרות אינסוף פעמים

$C=\phi$ – *compassion* - באופן דומה לדרישת ה – *justice*

<u>שאלה 2:</u>

כדי להגדיר את תנאי הניצחון של לוח הסוקובאן, השתמשנו ב *LTLSPEC*.

כתבנו $LTLSPEC$ הבודק האם לא קיים לבסוף מצב שבו אחד מהתאים שהוגדרו במצב ההתחלתי של הלוח להיות שייכים לקבוצת ה -

$$goal\_states = \{.,+,*\}$$

יהיה *. בצורה זו, כאשר הלוח פתיר, נקבל את הפתרון כהפרכה, וכאשר הלוח לא פתיר, ה $nuXmv$ יחזיר $True$, ובפייתון נדפיס שהלוח לא פתיר.

לדוגמה, אם ישנן שלוש מטרות בתאים $[1][2], [2][1], [3][3]$, שבמצב ההתחלתי הן שייכות ל – $goal\_states$ (הן יכולות להיות בכל אחד מהמצבים של קבוצה זו במצב ההתחלתי כמובן), אזי תנאי הניצחון יהיה:

```
LTLSPEC !(F((SokobanBoard[1][2] = asterisk) & (SokobanBoard[2][1] = asterisk) &
(SokobanBoard[3][3] = asterisk)
```

חלק שני

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.
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.

שאלה 1

הערות כלליות על מבנה הקוד:

את כל הקודים בעבודה זו רשמנו בפייתון. הקוד רשום בארבעה קבצים סה"כ: main.py, SokobanBoardGenerator.py, SokobanBoardSolver.py, SokobanIterativeSolver

לפני כל הרצה של הקוד, צריך להזין בתוך הקובץ main.py את ה PATH למיקום בו נמצא לוח הסוקובאן בקובץ txt., ואת ה PATH למיקום בו יווצרו קבצי ה smv. וה out.

```
inputFilePath = r"C:\Users\Lenovo\Documents\FVS\boards\board4.txt"##############YOUR FILE HERE##############
modelFilePath = r"C:\Users\Lenovo\Documents\FVS\outputFiles\part4\board4.smv"##############YOUR PATH TO THE SMV MOODEL HERE##############
```

דוגמה ללוח סוקובאן שיכול להינתן כקלט בתור קובץ txt.:

```
1  #######
2  ###.###
3  ###$###
4  #.$@$.#
5  ###$###
6  ###.###
7  #######
```

כעת נפרט עבור פתרון הקוד של חלק זה.

כדי להפעיל את החלק הזה בקוד, יש להריץ את קובץ ה main.py, ולבחור '2' כאשר תישאלו איזה חלק ברצונכם להריץ:

```
Hello, and welcome to the Sokoban solver!
Please choose the part you want to run,
valid answers are 2, 3 or 4: 2
```

בחלק זה, כתבנו קוד הקורא קובץ txt. המכיל לוח סוקובאן המורכב מסמלים, כפי שנתון בהסבר למטלה. הקוד קורא את הלוח, וכותב קובץ nuXmv לפתירת הלוח. לאחר מכן הקוד יריץ את קובץ ה smv. שפותר את הלוח, ובמידה והלוח פתיר, הקוד ידפיס את אסטרטגיית הניצחון. אם הלוח לא פתיר- הקוד ידפיס שהלוח לא פתיר.

<u>שלב ראשון: קריאת הקובץ</u>

חלק זה של הקוד כתוב בתוך הפונקציה:

createSmvBoardFile(inputFilePath, modelFilePath)

הקוד יקרא את לוח הסוקובאן הנתון בתוך inputFilePath, ויהפוך אותו בתוך הפייתון למשתנה מסוג SokobanBoard שהגדרנו. הclass של ה SokobanBoard יבצע שלושה משימות עיקריות:

1. אתחול כל המשתנים ב NuXmv לפי הלוח הנתון. יכנס לתוך המשתנה: self.InitialBoardString
2. זיהוי כל התאים מסוג מטרה, כלומר כל התאים בקבוצה $goal\_states = \{.,+,*\}$, ושימוש בזה כדי להגדיר את תנאי הניצחון בלוח הסוקובאן לפי הלוח הנתון. יכנס לתוך המשתנה: *self.winConditions*
3. עבור כל תא בלוח, יגדיר את פונקציית המעברים המתאימה. בשלב הזה, הקוד יסנן מעברים מיותרים ולא יכתוב אותם ב nuXmv. לדוגמה- תא מסוג רצפה שמימינו נמצא תא מסוג קיר, הפייתון ימנע מלכתוב ל NuXmv לבדוק את המקרה בו השומר נמצא על הלוח, והתנועה תהיה תנועה ימינה – כיון שהמקרה הזה תמיד לא יהיה אפשרי, ולכן ניתן להכליל את המקרה הזה עם המקרים האחרים בהם המעבר יהיה מעבר idle, כלומר rho_i. יכנס לתוך המשתנה: self.transitionRelations

יתר המקרים הקבועים, כמו לדוגמה האתחול ופונקציית המעברים של המשתנה המתאר את התנועה, כתוב בצורה מפורשת בפייתון, ולא תלוי בלוח מסויים.

לאחר מכן, הקוד ישרשר את כל הנתונים האלה לסטרינג אחד ארוך בפורמט המתאים לקובץ nuXmv, וירשום אותו לתוך קובץ smv. חדש שהוא יצור.

<u>שלב שני: הרצה של קובץ ה nuXmv</u>

חלק זה של הקוד כתוב בתוך הפונקציה:

outputFilePath = run_nuxmv(modelFilename, engineType = None, steps = None)

בחלק זה, הקוד יקבל את מיקום קובץ המודל modelFilePath, כלומר מיקום הקובץ של ה smv., ויריץ את הקוד. בנוסף, יש אפשרות להכניס את סוג המנוע הרצוי להרצה של הקוד. במידה ונבחר מנוע מסוג "SAT", ניתן לבחור לחסום את כמות הצעדים המקסימלי. את כל הפלט של הרצת ה

NuXmv הקוד ישמור בתוך קובץ באותה תיקייה של ה modelFilePath, עם אותו השם בסיומת out.. הפונקציה תחזיר את מיקום הקובץ לתוך המשתנה outputFilePath.

<u>שלב שלישי: הדפסת תוצאת ההרצה</u>

חלק זה של הקוד רשום בתוך הפונקציה:
get_board_result(outputFilePath)

בחלק זה, הקוד עובר על כל ההדפסות של ה NuXmv שנמצאות בקובץ outputFilePath, וקובע האם ה nuXmv מצא פתרון ללוח, ואם כן- ידפיס את הפתרון בפורמט של התנועות המהוות את אסטרטגיית הניצחון. אחרת- הקוד ידפיס שהלוח לא פתיר.

לפי האופן בו הגדרנו את תנאי הניצחון, אם הלוח פתיר- אזי ה nuXmv יכתוב כפלט שתנאי ה LTL שלנו שגוי (FALSE) ויצליח למצוא אסטרטגיית ניצחון, המביאה למצב שבו על כל התאים מסוג מטרה (כלומר התאים שבהתחלה נמצאו בתוך הקבוצה $goal\_states = \{.,+,*\}$) הופכים להיות תאים מסוג קופסא על מטרה (*). אם הלוח לא פתיר, ה nuXmv יכתוב כפלט שתנאי ה LTL שלנו נכון (TRUE). לכן הקוד יקרא את קובץ הפלט של ה nuXmv, ויבדוק האם ה nuXmv קבע שתנאי ה LTL הוא TRUE או FALSE. אם התוצאה היא TRUE – הקוד ידפיס שהלוח לא פתיר. אחרת- הקוד יבדוק מהם כל התאים שצריכים להפוך ל (*), לפי הדפסת תנאי ה LTL. בכל state חדש בפלט של ה nuXmv, הקוד יזהה את התנועה הנוכחית שנעשתה. כאשר הקוד יזהה שכל התאים מסוג מטרה, הפכו להיות תאים מסוג קופסא על מטרה – הוא יפסיק לקרוא את הקובץ פלט (.out), וידפיס את אסטרטגיית הניצחון. הסיבה לכך היא שה nuXmv נותן סדרת מצבים שמביאים למצב שבו כל תאי המטרה הופכים לתאים של קופסא על מטרה, אך לעיתים הלולאה תכלול בסוף תנועות שאינם חלק מאסטרטגיית פתירת לוח הסוקובאן, כיון שהפתרון של ה nuXmv הוא פתרון אינסופי. קוד הפייתון יפסיק כאשר הלוח מגיע למצב של פתרון. בנוסף, הפייתון יסנן תנועות אפשריות שאינן חלק מאסטרטגיית הניצחון. לדוגמה, אם ה NuXmv ייתן פתרון שכולל תנועה לכיוון מסוים שתיתן לכל התאים מקרה של מעבר rho_i כמו ניסיון ללכת ימינה מתא מסויים, כאשר מימין לתא יש קיר. תנועה זו אפשרית לפי המודל שבנינו, אך לא מהווה חלק מאסטרטגיית הניצחון.

<u>שאלה 2</u>

פתרונות מפורטים לחלק זה נמצאים בגיטהב במיקום: outputFiles/part2

בחלק זה, ניתן למצוא את קבצי ה smv, .out. עבור כל לוח כפתרון לשאלה זו.

כל הפקודות שבהן השתמשנו הורצו כחלק מהפונקציה run_nuxmv שעליה הוסבר בחלק הקודם.

עבור הרצה ללא בחירה מנוע כלשהו השתמשנו בפקודה הבאה (רלוונטי לחלק זה, חלק 2):

```
nuXmv.exe modelFilename
```

כאשר `modelFilename` מציין את שם מודל ה - smv הרלוונטי.

עבור הרצה במנוע BDD השתמשנו בפקודה הבאות אחת אחרי השנייה (מכאן והלאה זה רלוונטי לחלק 3):

```
nuXmv.exe -int modelFilename
go
check_ltlspec
quit
```

כאשר `modelFilename` מציין את שם מודל ה - smv הרלוונטי.

עבור הרצה במנוע SAT השתמשנו בפקודות הבאות אחת אחרי השנייה:

```
nuXmv.exe -int modelFilename
go_bmc
check_ltlspec_bmc
quit
```

ובמקרים בהם נרצה להריץ את המנוע הזה עם מספר מסוים של צעדים נשתמש ברצף הפקודות הבאות:

```
nuXmv.exe -int modelFilename
go_bmc
check_ltlspec_bmc -k steps
quit
```

כאשר `modelFilename` מציין את שם מודל ה - smv הרלוונטי, וה - `steps` מציין את מספר המקסימלי עבור הפתרון.

שאלה 3

פתרונות מפורטים לחלק זה נמצאים בגיטהב במיקום: outputFiles/part2

בחלק זה, ניתן למצוא את קבצי ה jpg. עבור כל לוח כפתרון לשאלה זו.

דוגמא להדפסה ללוח שאינו פתיר:

```
Hello, and welcome to the Sokoban solver!
smv model that have been created from C:\Users\אורה\FVS\boards\board1.txt saved at C:\Users\אורה\FVS\outputFiles\part2\board1.smv
The output of running Nuxmv on C:\Users\אורה\FVS\outputFiles\part2\board1.smv saved to C:\Users\אורה\FVS\outputFiles\part2\board1.out
*******************************************
********    NOT SOLVABLE    ***************
*******************************************
```

דוגמה להדפסה ללוח פתיר:

```
Hello, and welcome to the Sokoban solver!
smv model that have been created from C:\Users\אורה\FVS\boards\board4.txt saved at C:\Users\אורה\FVS\outputFiles\part2\board4.smv
The output of running Nuxmv on C:\Users\אורה\FVS\outputFiles\part2\board4.smv saved to C:\Users\אורה\FVS\outputFiles\part2\board4.out
*******************************************
********    SOLVABLE    *******************
*******************************************
The wining strategy for the keeper is:
*******************************************
*    L-R-U-D-R-L-D    *
*******************************************
```

נשים לב כי עבור ההדפסה של הפתרון, לקחנו רק את התנועות הרלוונטיות לפתרון. כך לדוגמה, אם במצב ההתחלתי התנועה מוגדרת להיות ימינה, אך מימין לשחקן יש קיר (ולכן התנועה לא אפשרית, ושום תא בלוח לא משתנה, אלא רק התנועה תשתנה במצב הבא), לא ניקח את התנועה ימינה כחלק מהפתרון (כי היא לא חלק מהפתרון עבור לוח הסוקובאן, אף על פי שזהו מצב אפשרי עבור המודל שנבנה עבור ה – smv), אלא ההצגה של הפתרון מתחילה ומסתיימת רק במסלול התקין של הפתרון, כמו שדרוש.

חלק שלישי

Part 3

1. Measure performance of nuXmv's BDD and SAT Solver engines on each of the models.
2. Compare the performance of the two engines. Is one engine more efficient than the other?

שאלה 1

פתרונות מפורטים לחלק זה נמצאים בגיטהאב במיקום: outputFiles/part3

פלט זמני הריצה כפי שמתקבל מהפיתון, נשמר בתוך קבצי time., כאשר השם של כל קובץ מציין איזה לוח נבדק, ובאיזה מנוע. בחלק זה, הרצנו לוחות סוקובאן פתירים בלבד.

לפני הרצת הקוד, עבור חלק זה ניתן להזין בתוך הקובץ main.py את מספר הצעדים המקסימלי להרצה במנוע "SAT". בנוסף, ניתן להזין את מספר הפעמים הרצוי שבו ה nuXmv יריץ את הלוח (הסבר מפורט יותר על מהות המשתנה הזה ינתן בהמשך)

```
steps=None ###############Optional: Enter a selected number of steps here (defualt is None)##########
avergeTime = 0
iters = 10 #######################optional: insert HERE number######################
```

כדי להפעיל את החלק הזה בקוד, יש להריץ את קובץ ה main.py, ולבחור '3' כאשר תישאלו איזה חלק ברצונכם להריץ:

```
Hello, and welcome to the Sokoban solver!
Please choose the part you want to run,
valid answers are 2, 3 or 4: 3
```

לאחר מכן, תישאלו באיזה מנוע תרצו שה nuXmv ירוץ, עליכם להכניס "SAT" או "BDD". אם סופק לפני הרצת הקוד, אזי בבחירת מנוע SAT, מספר הצעדים יהיה חסום ע"י הערך במשתנה "steps".

```
Please enter which type of engine to run,
valid answers are SAT or BDD:
```

הקוד יקבל כקלט לוח סוקובאן בקובץ txt. ויבנה קובץ smv. מתאים שינסה לפתור את הלוח, כמו בחלק הקודם. לאחר מכן, הקוד יריץ את קובץ ה nuXmv מספר פעמים, וימדוד בכל פעם את הזמן שלוקח להגיע לפתרון. הקוד יבצע ממוצע על זמני הריצה, וידפיס את הזמן הממוצע שלקח להריץ את הלוח המסויים, עבור המנוע שנבחר.

שלב ראשון: קריאת הקובץ

מתבצע בדיוק כמו בחלק הקודם

שלב שני: הרצה של קובץ ה nuXmv ומדידת זמנים

בשלב זה, הקוד יבקש לקבל כקלט באיזה מנוע תרצו להריץ את קובץ ה smv.:

```
Please enter which type of engine to run,
valid answers are SAT or BDD:
```

עבור הרצה במנוע SAT, ניתן להכניס לפני הרצת הקוד את מספר הצעדים המקסימלי לריצה במשתנה "steps" בקובץ main.py.

הקוד רץ בלולאה לפי מספר הפעמים הרשום במשתנה "iter". כרגע המשתנה מוגדר להיות 10, אך ניתן לשנות זאת לפני ההרצה של הקוד, בקובץ main.py.

בכל איטרציה של הלולאה, הקוד יקרא לפונקציה

MeasureRunTime(modelFilePath, engineType, steps)

הפונקציה מודדת את נקודת הזמן בכניסה, לאחר מכן קוראת לפונקציה
run_nuxmv(modelFilename, engineType, steps)

שמריצה את קובץ המודל smv. כפי שהוסבר בחלק הקודם, ואז היא מודדת את נקודת הזמן בסיום. הפונקציה MeasureRunTime מחזירה את זמן הריצה הכולל של הרצת קובץ ה smv. עבור הלוח המסוים, וכן את מיקום קובץ הפלט של ההרצה (.out). נדפיס את זמן הריצה של האיטרציה הנוכחית. בסוף ריצת כל האיטרציות, הפונקציה מדפיסה את הזמן הממוצע שלקח להרצה של הלוח.

מטרת החזרה ומציאת הממוצע היא לתת תמונת מצב נכונה עבור כמות הזמן שלוקח להריץ את הלוח, וממוצע על מספר ריצות הוא מדויק יותר מאשר הרצה ספציפית.

שאלה 2

מהתבוננות בתוצאות הריצה, ניתן לראות שבלוחות הבאים:

Board2, board4, board11, Board8, board10

אין פער ניכר בזמני הריצה בין המנועים השונים. לוחות אלו הם קטנים ופשוטים לפתרון בזמן ריצה קצר.

ניתן לראות כי ללוחות: Board2, board4, Board8 יש הטיה קלה בזמן הריצה לטובת מנוע ה BDD.
לעומת זאת ללוחות: board10, board11 יש הטיה קלה בזמן הריצה לטובת ה SAT.

עבור לוח: Board12

מתקבל כי מנוע ה – BDD יותר מהיר, ולעומת זאת, בלוחות הבאים:

board9, ,Board7

מתקבל כי מנוע ה – SAT יותר מהיר.

מהתבוננות כלפי חוץ על הלוחות הנ"ל, ניתן להציע כי בלוחות בהם מנוע ה BDD הצליח להגיע מהר יותר לתוצאה, מדובר בלוחות בהם התנועה היתה מוגבלת. כלומר, מכיוון שהיו הרבה "קירות" במרכז הלוח, עבור הרבה מהתאים, פונקציית המעברים היתה קטנה יותר. לעומת זאת בלוחות בהם מנוע ה SAT הצליח להגיע מהר יותר לתוצאה, היו יותר תאי לוח שפונקציית המעברים שלהם היתה שלמה, כלומר- היו יותר אופציות לבדוק בכל צעד.

בנוסף, ניתן לראות שגם הלוח המסובך ביותר, לוח 12, לקח בממוצע פחות משעה (בכל אחד מהמנועים), אף על פי שהוא לוח גדול שמכיל 136 תאים (8*17). ניתן לייחס את הריצה המהירה הזו לעובדה שבנינו בצורה חכמה את מודלי ה – smv על ידי סינון מצבים מיותרים וכלילתם כחלק מפונקצייה המעברים rho_i, וכפי שהסברנו בהרחבה בחלק הראשון של הדו"ח (בהסבר לגבי חלק 2 במטלה). סינון זה מוביל לחסכון של בדיקות מיותרות, וזה גורם לחסכון משמעותי בזימן הריצה של ה- nuXmv.

חלק רביעי

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.
2. Indicate runtime for each iteration, as well as the total number of iterations needed for a given board.
3. Test your iterative solution on larger more complex Sokoban boards.

שאלות 1,2

פתרונות מפורטים לחלק זה נמצאים בגיטהאב במיקום: outputFiles/part4

קבצי הפלט עבור חלק זה מחולקים למספר רמות אבסטרקציה:

1) תקיות עבור הלוח אותו פתרנו, המכילות:
2) תקיות עבור כמה קופסאות ה nuXmv ניסה לפתור בכל איטרציה המכילות:
3) קבצי ה smv. ו out. עבור ריצה של כל איטרציה. בנוסף בתוך התקיה יש קובץ iterative.

הקובץ הזה מכיל את פלט ההרצאה של חלק 4 עבור הלוח הנתון ומספר הקופסאות לפתרון, בפרט ניתן למצוא שם אילו מטרות נפתרו בכל איטרציה, ואת זמן הריצה.

לפני הרצת הקוד, עבור חלק זה צריך להזין בתוך הקובץ main.py את מספר הקופסאות אותם נרצה לפתור בכל איטרציה

```
engineType = getEngine()
numBoxes = 2  ######################insert HERE number for the amount of boxes per iteration#####################
board = createBoard(inputFilePath)
```

כדי להפעיל את החלק הזה בקוד, יש להריץ את קובץ ה main.py, ולבחור '4' כאשר תישאלו איזה חלק ברצונכם להריץ:

```
Hello, and welcome to the Sokoban solver!
Please choose the part you want to run,
valid answers are 2, 3 or 4: 4
```

לאחר מכן, תישאלו באיזה מנוע תרצו שה nuXmv ירוץ, עליכם להכניס "SAT" או "BDD". עבור הרצה עם מנוע SAT, ניתן להכניס באופן ידני לפני הריצה את מספר הצעדים המקסימלי להרצה בתוך המשתנה steps בקובץ main.py

```
Please enter which type of engine to run,
valid answers are SAT or BDD:
```

הקוד יקבל כקלט לוח סוקובאן בקובץ txt., יבחר את המטרות אותם הוא רוצה לפתור באיטרציה הנוכחית, ויבנה קובץ smv. מתאים שינסה לפתור את הלוח עבור המטרות האלו. לאחר מכן, הקוד יריץ את קובץ ה nuXmv וימדוד את הזמן שלוקח להגיע לפתרון. אם האיטרציה הנוכחית היתה פתירה – הקוד יבחר מטרות חדשות אותם הוא רוצה לפתור, יבנה קובץ smv. מתאים ויפתור. אם

האיטרציה הנוכחית הייתה לא פתירה – הקוד ידפיס שהלוח לא פתיר, ויסיים את הריצה. נרוץ בלולאה עד שכל הלוח ייפתר.

שלב ראשון: קריאת הקובץ

באופן דומה לחלקים קודמים, הקובץ יקרא את לוח הסוקובאן, אך כאן הוא רק יעדכן את המשתנה בפייתון מסוג הclass של ה SokobanBoard, בלי ליצור את קובץ ה smv. עדיין.

שלב שני: מציאת כל המטרות בלוח:

בשלב זה הקוד ירוץ על ה sokobanBoard שנבנה בהתאם ללוח הסוקובאן שהתקבל כקלט, וימצא את מיקומי המטרות של כל הלוח. הפונקציה תקבל לתוך המשתנה boardGoals את המיקומים הללו.

boardGoals = getGoalsLocs(board)

שלב שלישי: פתרון איטרטיבי של הלוח

בשלב זה הקוד ירוץ בלולאה עד לפתרון כל המטרות הקיימות בלוח. חלק זה מורכב משלושה שלבים עיקריים:

1) בחירת מטרות לפתרון
2) עדכון של הלוח לאיטרציה הנוכחית
3) הרצה של nuXmv ומדידת זמני ריצה

חלק זה רץ עם קריאה לפונקציה:

sampleNCreateBoards(Ipath, Opath, boardGoals, numBoxes, board, engineType)

כאשר הפונקציה מקבלת כקלט את מיקום לוח הסוקובאן, המיקום בו יווצרו קבצי ה smv. וה out., מיקומי המטרות בלוח, מספר הקופסאות שנרצה שהקוד יפתור בכל איטרציה. בנוסף מקבל משתנה board מסוג SokobanBoard וכן באיזה סוג מנוע נרצה שה nuXmv יפתור את הלוח.

נפרט על כל שלב:

1) בחירת מטרות לפתרון – בשלב זה מתבצעת בחירה של המטרות אותם ה nuXmv יפתור באיטרציה הנוכחית. אם מספר המטרות שעוד נותרו בלתי פתורות גדול ממספר המטרות אותם נרצה לפתור בכל איטרציה, נבחר באופן אקראי מטרות לפתרון, כמספר המטרות שנדרש לפתור בכל איטרציה. כלומר אם נרצה לפתור מטרה אחת בכל איטרציה, ונותרו 3 מטרות בלתי פתורות- נבחר באופן אקראי מטרה אחת שאותה נפתור באיטרציה הזאת. אחרת, נבחר את כל המטרות שנותרו לא פתורות בלוח. בחרנו בצורה רנדומלית, ולא לפי מטרה שקרובה לקופסא מסוימת או לשחקן, שכן לכל אחת מהבחירות הללו יש דוגמה נגדית שתגרום לפתרון לא אופטימלי ולכן העדפנו את הבחירה הרנדומלית. כמו כן, בחירת מטרות בצורה רנדומלית יכולה לעזור לנו לאחר מכן לבחור את המטרות בצורה כזו שתתן פתרון כמה שיותר יעיל במקרים בהם אין לנו הנחה או השערה כיצד לבחור את המטרות כדי לייצר את הפתרון המהיר ביותר.

2) עדכון של הלוח לאיטרציה הנוכחית – בשלב זה ניצור קובץ .smv המתאים לפתרון הלוח באיטרציה זו, באמצעות קריאה לפונקציה:
createSmvBoardFileIteration(Ipath, Opath, iterationGoals, numBoxes, board = None, iteration = 1, smvSolution = None)

פונקציה זו מקבלת כקלט את מיקום לוח הסוקובאן המקורי, מיקום רצוי ליצור בו את קובץ ה .smv המתאים לאיטרציה זאת, מיקומי המטרות אותם נרצה לפתור באיטרציה הזאת, מספר המטרות לפתרון בכל איטרציה (מיועד לקביעת שם קובץ ה .smv), מצב הלוח בתחילת האיטרציה הקודמת (או המצב ההתחלתי של הלוח, במקרה של האיטרציה הראשונה), מספר האיטרציה הנוכחית, ולבסוף מקבל את תוצאת ההרצה של קובץ ה .smv באיטרציה הקודמת.

הפונקציה מגדירה את מיקומי תאי המטרה של האיטרציה הנוכחית בפייתון במשתנה מסוג הclass של ה SokobanBoard, באופן דומה לחלקים הקודמים, ומכניסה לתוך המשתנה *self.winConditions*. בכל איטרציה הוספנו לתנאי הניצחון את הקופסאות שנוספו באיטרציה זו בנוסף לקופסאות שנפתרו באיטרציה הקודמת (שכן הן כבר נפתרו, ולא נרצה להגיע למצב שבו בסוף כל האיטרציות חלק מהמטרות לא פתורות, או למצב שבו מטרות שכבר נפתרו באיטרציה מסוימת ואמורות להיות פתורות, יהיו לא פתורות באחד מהאיטרציות הבאות.). **כאמור, השתמשנו באותה לוגיקה טמפורלית בדומה לחלקים הקודמים, רק שכל איטרציה הוספנו לה עוד מטרות. כלומר הלוגיקה הטמפורלית באיטרציה מסוימת תיראה כך:**

;LTLSPEC !(F((SokobanBoard[i][j] = asterisk) & (SokobanBoard[n][m] = asterisk) & ...))

בהתאם למה שכתבנו לעיל, באיטרציות הבאות נוסיף לה מטרות שלא היו קיימות בתוך הסוגריים באיטרציות הקודמות, עד שנגיע לאיטרציה הסופית בה אנו פותרים עבור כל המטרות.

כך לדוגמה, אם היו לנו 4 מטרות בסה"כ בלוח, ובאיטרציה הראשונה פתרנו עבור 2 קופסאות (המיקומים נבחרו רק לשם הדוגמה):

;LTLSPEC !(F((SokobanBoard[2][8] = asterisk) & (SokobanBoard[1][9] = asterisk)))

אזי באיטרציה הבאה תנאי הניצחון יהיה רלוונטי לכל ארבעת המטרות (גם כאן, המיקומים נבחרו רק לשם הדוגמה):

LTLSPEC !(F((SokobanBoard[2][8] = asterisk) & (SokobanBoard[1][9] = asterisk) & (SokobanBoard[5][5] = asterisk) & (SokobanBoard[3][4] = asterisk)));

לאחר מכן, הפונקציה משתמשת בקובץ ה .out של האיטרציה הקודמת על מנת לעדכן את מצב הלוח בסוף האיטרציה הקודמת- כלומר קובע מהו המצב ההתחלתי של הלוח באיטרציה הזאת באמצעות הפונקציה: update(smvSolution, board)

לשם כך, הפונקציה עוברת על קובץ ה .out באופן דומה לפונקציה get_board_result(outputFilePath), ועבור כל שינוי של תא בלוח היא מעדכנת את המשתנה של תא הלוח של board בערך החדש שלו. כמו בפונקציית get_board_result, כאשר הפונקצייה מגיעה ל state שבו כל המטרות הרצויות אכן הגיעו למצב של קופסא-על-מטרה, הוא מפסיק לרוץ על ה .out, ומחזיר את הלוח המעודכן.

לבסוף, כמו בחלקים הקודמים, הפונקציה משרשרת את כל הנתונים לפורמט המתאים לקובץ .smv ויוצר קובץ חדש מסוג .smv המתאים לפתור את האיטרציה הנוכחית. נשים לב שיתר הפרטים בקובץ נשארים זהים בין האיטרציות (כל פונקציית המעברים וכו')

יש לשים לב כי עבור כל איטרציה, הקוד ישרשר לשם של קבצי הפלט את מספר הקופסאות שנפתרו באיטרציה הזאת, ומה מספר האיטרציה.

לדוגמה עבור מיקום פלט נתון: "board4.smv"

מספר קופסאות שהקוד פותר בכל איטרציה: numBoxes = 2

עבור איטרציה מספר 1.

נקבל קבצי פלט עם השם:

board4_2_boxes_IterationModel_iter1.out ,

board4_2_boxes_IterationModel_iter1.smv

3) הרצה של nuXmv ומדידת זמני ריצה- בשלב זה נריץ את קובץ ה smv. ונמדוד את זמני הריצה שלו כמו בחלק 3 באמצעות הפונקציה
MeasureRunTime(modelFilePath, engineType, steps)
לבסוף, נבדוק שהריצה של ה smv. מצאה שהלוח פתיר. אם לא- נחזיר שהלוח לא פתיר, ונסיים את הריצה.

<u>שאלה 3</u>

כעת נדון בתוצאות זמן הריצה שקיבלנו עבור כל אחד מהלוחות. עבור כל לוח, ביצענו הרצה של כל האופציות האפשריות עבור מספר מסוים של מטרות בכל איטרציה. כך למשל, אם יש בלוח 4 מטרות, ניתן לבצע בכל איטרציה פתרון עבור מטרה אחת / עבור 2 מטרות / עבור 3 מטרות (כאשר במקרה שבוחרים במספר שלא מתחלק במספר המטרות הכולל, באיטרציה האחרונה תתבצע שארית המטרות, כלומר בדוגמה שלנו אם נבחר לפתור 3 מטרות בכל איטרציה, באיטרציה הראשונה נפתור את הלוח עבור 3 מטרות, ובאיטרציה השנייה נפתור את הלוח עבור כל ארבעת המטרות, כלומר עבור המטרה הבודדת שנותרה). כאמור, אנחנו ביצענו את כל האופציות עבור האפשריות עבור כל מספר מטרות בכל איטרציה. אם נמשיך עם הדוגמה לעיל, עבור לוח עם מטרות 4, כמו לוח 8, קיימות 3 תתי – תיקיות שמכילות כל אחת את הריצה האיטרטיביות עבור המספר הנבחר של המטרות לפתרון בכל איטרציה (1, 2 או 3). תתי התיקייה נקראות:

1_box_per_iter, 2_box_per_iter, 3_box_per_iter

וכן על זה הדרך בשאר הלוחות והאופציות האפשריות.

הרצנו כל אחד מהלוחות עם המנוע המהיר יותר עבורו, לפי התוצאות שקיבלנו בחלק 3. כעת נתבונן בזמני הריצה השונים שקיבלנו בריצות האיטרטיביות ונשווה אותן לתוצאות הממוצעות של זמן הריצה שקיבלנו בחלק 3. בחלק זה הרצנו בצורה איטרטיבית רק לוחות פתירים, ושיש בהם יותר ממטרה אחת. בניגוד לחלק הקודם, כאן לא ביצענו ממוצע על פני כמה ניסיונות עבר כל מספר מסוים של מטרות, כי זה יקח המון זמן ומשאבי חישוב שאין לנו (בערך פי 4 מהרצת החלק הקודם, שבמצטבר לקח להריץ כ 30 שעות), ובנוסף שימוש במחשב במהלך הרצת ה- nuXmv, פוגע בביצועים שלו ואנחנו צריכים את המחשב לעוד דברים.

בלוחות הקטנים והפשוטים ביותר, לוחות 4 ו – 11, ניתן לראות שזמן הריצה עבור כל איטרציה דומה לזמן הריצה הממוצע מחלק 3 עבור הפתרון של כל המטרות בפעם אחת. ניתן להסביר זאת בעובדה שכיוון שאילו לוחות קטנים ופשוטים, החלוקה לאיטרציות לא עוזרות ל – nuXmv לפתור אותם יותר מהר. כיוון שאם שזמן הריצה הכולל של הריצה האיטרטיבית בכל אחת מהריצות הללו הינו זמן הריצה הכולל של כל אחת מהאיטרציות, נקבל שזמן הריצה הכולל של הריצה האיטרטיבית הינו פחות או יותר כפולה של זמן הריצה הממוצע מחלק 3 (המספר שבו כופלים תלוי במספר המטרות שנפתר בכל איטרציה, שכן פרמטר זה קובע את מספר האיטרציות), ולכן במקרים כגון אלו החלוקה לאיטרציות אינה יעילה מבחינת זמן ריצה.

בנוסף, גם בלוח 7 (שהינו גדול ומסובך יותר מהלוחות 4,8), הריצה האיטרטיבית לא יעילה מבחינת זמן ריצה, וכפי שניתן לראות בתוצאות. הסיבה לכך היא שאחת מהמטרות הללו כבר נפתרה עבור המצב ההתחלתי של הלוח, ולכן עבור הפתרון של מטרות זו באחת משתי האיטרציות (בלוח יש בסך הכל 2 מטרות), ה – nuXmv בונה איזשהו לופ שפותר עבור מטרה זו. הלופ הזה לא באמת נצרך ודורש זמן ריצה. ניתן לראות שבאיטרציה שבה פותרים עבור המטרה שעוד לא נפתרה, זמן הריצה דומה לזמן הריצה הממוצע עבור לוח זה מחלק 3. כלומר בסך הכל נקבל סכום של זמן ריצה שדומה לזמן הריצה הממוצע מחלק 3, ועוד זמן ריצה של לופ חסר משמעות, מה שייצור זמן ריצה גדול יותר מזמן הריצה שחושב בחלק 3.

עוד לוח שבו הריצה האיטרטיבית יצאה לא יעילה הוא לוח 10. מניתוח של קבצי ה – out של הפתרונות האיטרטיביים, ניתן לראות כי הבחירה הרנדומלית של המטרה באיטרציה הראשונה בחרה קודם לפתור את המטרה הרחוקה יותר מבין שתי המטרות, דבר שהוביל לזמן ריצה ארוך. אך מנגד, גם אם היינו בוחרים לפתור קודם עבור המטרה הקרובה יותר, ניתן להעריך שהיינו מקבלים זמן ריצה דומה לזמן הריצה הממוצע, שכן באיטרציה השנייה עבור לוח זה אנחנו פותרים עבור המטרה

השנייה, שאנו קרובים אליה אפילו יותר מאשר במצב ההתחלתי של הלוח, ועדיין לפי קבצי הפלט ניתן לראות כי זמן הריצה באיטרציה השנייה קרוב לזמן הריצה הממוצע, כלומר בכל מקרה ניתן לומר שגם אם היינו מתחילים מהמטרה הקרובה יותר, היינו מקבלים זמן ריצה כולל (עבור כל האיטרציות) שהינו פי 2 מזמן הריצה הממוצע שחושב בחלק 3.

כעת נתבונן בלוחות 8,9, שבהן אנו מקבלים תוצאות שונות מהלוחות הקודמים שניתחנו בחלק זה. אמנם בלוחות אלו סכום זמן הריצה הכולל של כלל האיטרציות (בכל אחד מהאופציות עבור מספר מטרות מסוים בכל איטרציה) הינו גדול מזמן הריצה הממוצע שחושב בחלק 3 עבור לוחות אלו, אבל בחלק מהאיטרציות בלוחות אלו, ניתן לראות זמן ריצה נמוך יותר (לאיטרציה מסוימת) מזמן הריצה הממוצע מחלק 3. גם כאן, ניתן לייחס כל זמן ריצה שהיה נמוך יותר מזמן הריצה שחושב בחלק 3, לבחירה של המטרות שיפתרו באותה האיטרציה, שכן במקרים בהם זמן הריצה לאיטרציה נמוך יותר, מדובר בד"כ במקרים בהם נבחרו המטרות הקרובות יותר לשחקן, כפי שניתן לראות בלוחות ובקבצי ה – out וה – smv של האיטרציות.

הלוח המפתיע ביותר הינו לוח 12, שהינו הלוח הגדול והמסובך ביותר, שכולל בתוכו חמש מטרות, ומכיל 136 תאים (17*8). עבור לוח זה, עבור פתרון של מטרה בודדת בכל איטרציה, ועבור פתרון של 2 מטרות בכל איטרציה, קיבלנו זמן ריצה כולל שגדול מהזמן שחושב בכל חלק 3 (וגם כאן, מניתוח קבצי ה- out וה – smv ניתן לראות שבחירה לא יעילה של המטרות הובילה לזמן ריצה ארוכים לאיטרציה מסוימת). מנגד, עבור פתרון של 3 מטרות באיטרציה או 4 מטרות באיטרציה (והשארית נפתרה כמובן באיטרציה השנייה כנ"ל), קיבלנו זמן ריצה נמוך משמעותית מזמן הריצה שחושב בחלק 3. גם כאן, ניתן לראות שבחירה מוצלחת של מטרות הובילה לזמן ריצה נמוך משמעותית בכל איטרציה, דבר שהוביל לכך שסכום זמן הריצה הכולל של כל האיטרציות היה נמוך משמעותית מזמן הריצה הממוצע שחשוב בחלק 3 עבור לוח זה.

לאחר ניתוח התוצאות, ניתן היה לכאורה להסיק שאסטרטגיה מוצלחת שתוביל לפתרון איטרטיבי בזמן מהיר היא בחירה של מטרות שקרובות לשחקן, שכן זה מה שעבד טוב במקרים בהם קיבלנו זמן ריצה נמוך מזמן הריצה הממוצע שחושב בחלק 3. אמנם, מסקנה זו נכונה ביחס ללוחות הנתונים שעליהם הרצנו את ה – nuXmv, אך מנגד חשוב לזכור שרצינו לנסח אסטרטגיה כללית לבחירת מטרות. כיוון שעלולים להיות מקרים רבים של לוחות בהם המטרות שקרובות לשחקן רחוקות מאוד מהקופסאות שצריך לדחוף אליהן, אסטרטגיה זו לא יכולה להיות האסטרטגיה הרצויה במגוון רחב של מקרים, שכן היא עלולה להוביל לזמן ריצה ארוך מאוד. לכן, כפי האמור לעיל, בחרנו באסטרטגיה של בחירת מטרות רנדומלית, שיכולה לתת תוצאות טובות עבור מגוון רחב של מקרים שונים, ביניהם גם המקרים שבחנו וגם המקרים שלא בחנו במסגרת עבודה זו. זאת ועוד, הבחירה הרנדומלית היא בחירה טובה, כיוון שהיא יכולה לכוון אותנו כיצד לבחור את המטרות שיובילו לפתרון האיטרטיבי המהיר ביותר, במקרים בהם אין לנו אינטואיציה כיצד לבחור את המטרות.

לסיכום, בחלק זה ראינו את התוצאות של הריצה האיטרטיבית על הלוחות השונים, ובמקרים המסובכים יותר ראינו את ההשפעה של בחירת המטרות על זמן הריצה של ה – nuXmv. ראינו כי בחירה נכונה של המטרות שיפתרו באיטרציה מסוימת משפיעה באופן משמעותי על זמן הריצה, הן לטובה והן לרעה. לאור התוצאות והמסקנה שהסקנו, ניתן לומר שבנייה של אלגוריתם שיבחר את המטרות שיפתרו בכל איטרציה תיתן זמני ריצה יעילים ומהירים יותר ביחס לריצה על כל המטרות בבת אחת (כמו שנעשה בחלק 3). כמו שכתבנו לעיל, בחרנו בבחירה הרנדומלית של המטרות, שכן הבחירה במטרה הקרובה ביותר או בקופסא הקרובה ביותר אינה הדרך היעילה (זאת משום שדוגמאות נגדיות פשוטות לכל בחירה של אחת משתי האופציות הללו יובילו לזמן ריצה ארוך), ולכן פיתוח האלגוריתם צריך להיות מורכב יותר (גם מהאופציה של בחירה רנדומלית של מטרות) ויכול להוות בסיס למחקר עתידי ומעמיק, שימשיך את התהליך של עבודה זו.