פרויקט מסכם

אימות פורמלי וסינתזה

**תאריך:** 19.05.2024

**מגישים :**

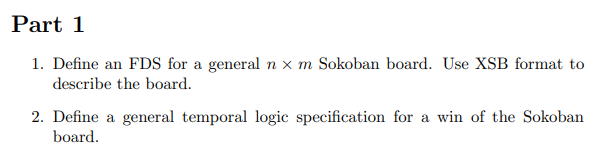
נועם דיאמנט 208520262  
אורה רחל ווצלר 212058689

**מבוא**

בפרויקט זה, נתבקשנו לכתוב תוכנית פייתון, המייצרת עבור לוח סוקובאן נתון בפורמט XSB, את מודל ה – smv המתאים לו, ומריצה את תוכנת ה – nuXmvעל 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

**חלק ראשון**

שאלה 1

נגדיר את ה - של לוח הסוקובאן:

– משתני המערכת – קיימים שני סוגי משתנים:

משתנים ששייכים ללוח הסוקובאן: משתנים שהם תאים בלוח, כאשר כל תא בלוח מהווה משתנה יחיד. נתאר את הלוח כמערך דו ממדי עם תאים. כל משתנה של תא בלוח יראה כך: , שהוא התא בשורה ה- ובעמודה ה – .

כל משתנה בלוח יכול לקבל את אחד מבין הערכים הבאים[[1]](#footnote-1):

בנוסף, יש לנו משתנה המתאר את תנועת השומר:

נגדיר זאת גם בצורה מתמטית:

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

*- 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 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 כלשהו הוא קיר ('#'), אזי הפייתון יציב לו בתור פונקציית מעבר את (פונקצייה זו היא פונקציית ה – 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, או שאין עליו כלום. לכן חילקנו את המצבים הנותרים (אחרי שטיפלנו באופציה של הקיר), לשתי קבוצות:

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 שלא נכתבו 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,  
 , הפייתון מראש יבדוק האם מדובר בתא שהבסיס שלו הוא רצפה ללא מטרה, ולכן בכל תנאי הוא יבדוק ויציב רק אחד מהאפשרויות המתאימות מתוך הקבוצה:

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

להלן מוצגת דוגמה כללית עבור מיקום כללי כלשהו בלוח סוקובאן נתון. יש לשים לב שבקבצים האמייתים מופיעים מספרים במקום 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 = – justice - המשחק אמור להסתיים לאחר מספר סופי של צעדים, ולכן אין מצב שאמור לקרות אינסוף פעמים*

*C= – compassion - באופן דומה לדרישת ה – justice*

*שאלה 2:*

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

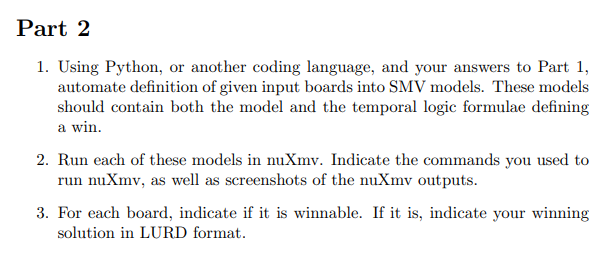
*כתבנו הבודק האם לא קיים לבסוף מצב שבו אחד מהתאים שהוגדרו במצב ההתחלתי של הלוח להיות שייכים לקבוצת ה -*

*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 , 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**

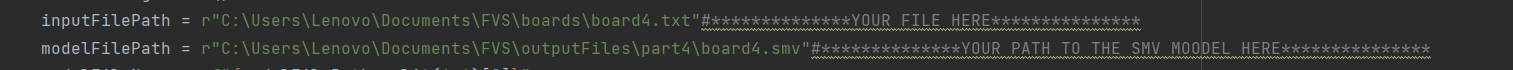


שאלה 1

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



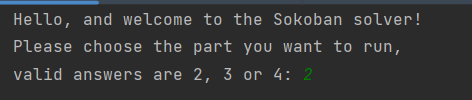
דוגמה ללוח סוקובאן שיכול להינתן כקלט בתור קובץ .txt:

תמונה שמכילה צילום מסך, גופן, טקסט, מספר

התיאור נוצר באופן אוטומטי

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:



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.

שלב ראשון: קריאת הקובץ

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 *, 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.

שלב שני: הרצה של קובץ ה nuXmv

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.

שלב שלישי: הדפסת תוצאת ההרצה

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 ) 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.

שאלה 2

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

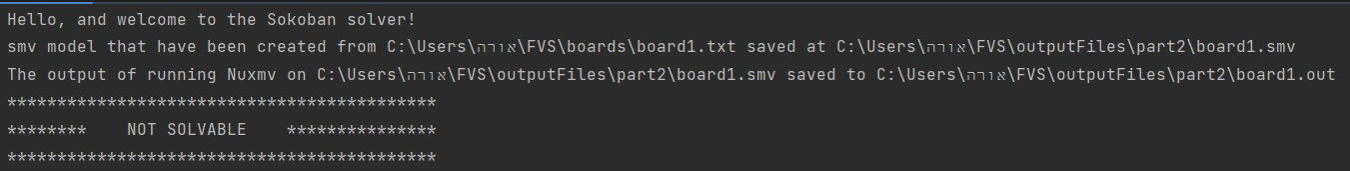
כאשר modelFilename מציין את שם מודל ה -smv הרלוונטי, וה - steps מציין את מספר המקסימלי עבור הפתרון.

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:



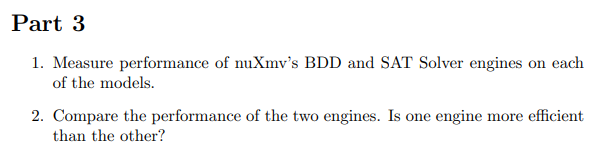
דוגמה להדפסה ללוח פתיר:

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

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**

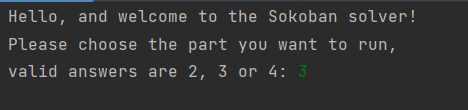


שאלות 1,2

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.

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



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



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:

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.

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:

Board7, board9,

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**

תמונה שמכילה טקסט, צילום מסך, גופן, אלגברה

התיאור נוצר באופן אוטומטי

שאלות 1,2,3

בחלק זה פתרנו את הלוחות בצורה איטרטיבית, פירוט מלא של המודלים האיטרטיביים וזמני הריצה שלהם נמצא בגיטהאב. דוגמה למודלים איטרטיביים תיראה כך: אם המודל המקורי (הלא איטרטיבי) יקרא כך: ccc.smv ובו ישנן 7 מטרות, ובכל איטרציה אנחנו רוצים לפתור 3 מטרות (למעט האיטרציה האחרונה שבה נפתור עבור המטרה האחרונה), המודלים האיטרטיביים יקראו כך:

ccc\_IterationModel\_iter1.smv

ccc\_IterationModel\_iter2.smv

ccc\_IterationModel\_iter3.smv

האופן בו פתרנו את הלוחות בצורה איטרטיבית:

בכל איטרציה בחרנו בצורה רנדומלית את המטרות אותן נפתור באיטרציה זו (לפי מספר האיטרציות שניתן כקלט לקוד, ובאיטרציה האחרונה פתרנו את שארית המטרות אם שארית המטרות קטנה ממספר המטרות בכל איטרציה). 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. לאחר בחירה זו, פתרנו את הלוח עבור אותן מטרות שנבחר, ביצענו עדכון ללוח לפי המצב הסופי של האיטרציה הקודמת, והמשכנו לפתרון האיטרציה הבאה, כאשר 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 מהקופסאות לא פתורות). 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)));

לכל איטרציה הדפסנו למסך את זמן הריצה שנדרש עבור איטרציה זו.

כמו כן, יתכנו מקרים בהם לאחר פתרון איטרציה אחת, הלוח כבר לא יהיה פתיר באיטרציות הבאות, ובמקרים הללו כאשר הגענו לאיטרציה לא פתירה הדפסנו שהלוח לא פתיר ועצרנו את התהליך.

דוגמאות ישנן בצורה מפורטת ובהרחבה בגיטהאב.

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. [↑](#footnote-ref-1)