פרויקט מסכם

אימות פורמלי וסינתזה

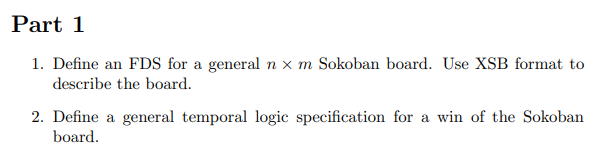
**תאריך:** 19.05.2024

**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.

**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.*

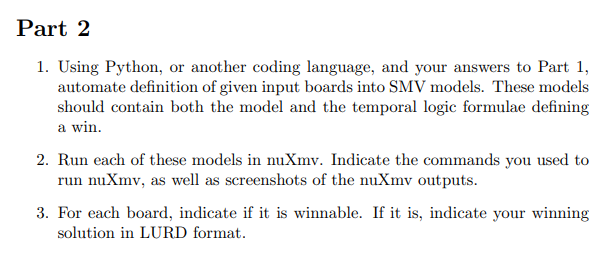
*כתבנו הבודק האם לא קיים לבסוף מצב שבו אחד מהתאים שהוגדרו במצב ההתחלתי של הלוח להיות שייכים לקבוצת ה -*

*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

חלק זה נמצא בצורה מפורטת בגיטהאב.

שאלה 2

גם חלק זה מפורט בקבצי הפייתון השונים וכמו שמפורט בגיטהאב.

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

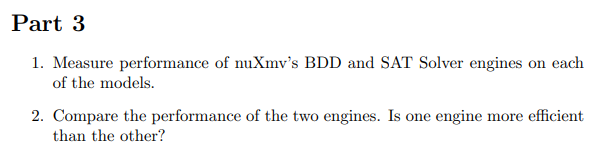
quit

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

שאלה 3

מפורט עבור כל לוח בגיטהאב. 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

הפתרונות לחלק זה מפורטים בגיטהאב.

ניתן לראות שבלוחות הבאים:

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

מתקבל כי מנוע ה – SAT יותר מהיר.

**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)