פרויקט מסכם

אימות פורמלי וסינתזה

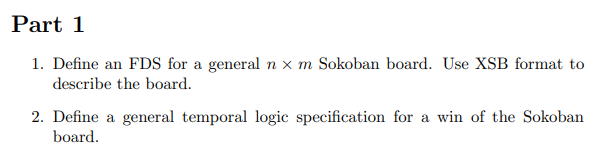
**תאריך:** 19.05.2024

**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, כאשר כתבנו את התוכנית שמייצרת את פונקציית המעברים בפייתון, בדקנו כבר בפייתון אלו מעברים אפשריים עבור כל אחת מהקבוצות הללו עבור הלוח הנתון (לדוגמה, אם מקום מסוים בלוח הוא שחקן, אך הוא רוצה לנוע ימינה ומימינו יש קיר, התנועה כמובן לא אפשרית. עוד דוגמה: לא ניתן לדחוף שתי קופסאות שממוקמות זו אחר זו, או לדחוף קופסא שצמודה לקיר לכיוון הקיר), וכל המעברים הלא אפשריים פשוט יוצבו ב- באופן ברירת מחדל עבור כל המצבים שלא נכתבו כאפשריים עבור אותו התא (כלומר התא ישאר במצבו הנוכחי). כלומר, אם הפייתון יזהה שתא סמוך, או תא בשכנות מדרגה 2, גורם לכך שמעבר מסוים אינו אפשרי, הוא לא יכתוב את התנאי ב nuXmv, מכיוון שזאת בדיקה מיותרת שתמיד תניב תוצאה שלילית. באופן זה, חסכנו זמן ריצה ב – nuXmv, כי לא בדקנו מלכתחילה מצבים שאינם אפשריים.

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.*

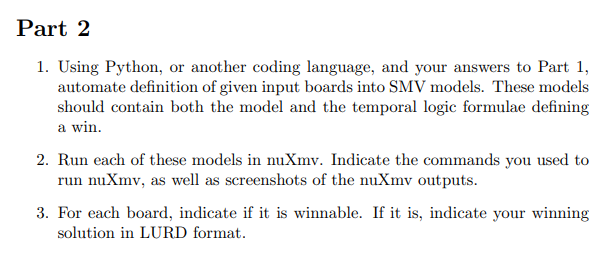
*כתבנו הבודק האם לא קיים לבסוף מצב שבו אחד מהתאים שהוגדרו במצב ההתחלתי של הלוח להיות שייכים לקבוצת ה -*

*יהיה \*. בצורה זו, כאשר הלוח פתיר, נקבל את הפתרון כהפרכה, וכאשר הלוח לא פתיר, נקבל שהלוח לא פתיר.*

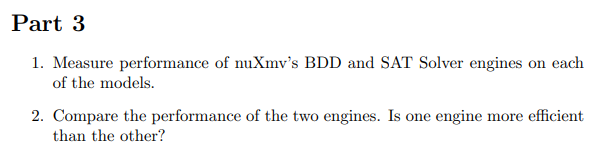
*לדוגמה, אם ישנן שלוש מטרות בתאים , שבמצב ההתחלתי הן שייכות ל – goal\_states (הן יכולות להיות בכל אחד מהמצבים של קבוצה זו במצב ההתחלתי כמובן), אזי תנאי הניצחון יהיה:*

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

**חלק שני**



**חלק שלישי**



**חלק רביעי**

תמונה שמכילה טקסט, צילום מסך, גופן, אלגברה

התיאור נוצר באופן אוטומטי

חלק ראשון:

1. נגדיר את ה FDS של לוח הסוקובן:

D = {V, , , J, C}

V – יש לנו n\*m משתנים שהם תאים בלוח, כאשר כל תא בלוח מהווה משתנה יחיד. נתאר את הלוח כמערך חד מימדי עם n\*m תאים. כל משתנה של תא בלוח יראה: board[i], יהיה התא בשורה (i/n) ובעמודה (i%n) כל משתנה בלוח לקבל אחד מבין הערכים הבאים:

{@, +, $, \*, #, ., -}[[2]](#footnote-2)

בנוסף, יש לנו משתנה המתאר את תנועת השומר:

- מצב התחלתי של המערכת – הלוח הנתון בתחילת כל משחק

*- המעברים האפשריים המערכת נתונים לפי המצבים הבאים:*

פונקציית המעברים של תנועת השומר היא לא דטרמיניסטית, ונתונה:

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

פונקציית המעברים של התאים בלוח:

הפונקציה נתונה עבור תא כלשהו. כאשר נריץ את הפייתון, הוא יציב את התא הספציפי בטבלה. כבר כאשר נריץ את הפייתון, נקבל שני פונקציות הצבה שונים.

אם הפייתון מזהה כי המצב ההתחלתי בלוח של תא [i][j] כלשהו הוא קיר ('#'), אזי הפייתון יציב לו בתור פונקציית מעבר את rho\_i, כיוון שהתא לעולם לא ישתנה:

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

אחרת, 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]…) , הפייתון מראש יבדוק האם מדובר בתא שהבסיס שלו הוא רצפה, 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 = {.,+,\*}

בנוסף, אם הפייתון יזהה שתא סמוך, או אם רלוונטי- תא בשכנות מדרגה 2, הוא קיר- הוא לא יכתוב את התנאי ב nuXmv, מכיוון שזאת בדיקה מיותרת שתמיד תניב תוצאה שלילית.

    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 = – המשחק אמור להסתיים לאחר מספר סופי של צעדים, ולכן אין מצב שאמור לקרות אינסוף פעמים*

*C= – באופן דומה לדרישת הצדק*

*שאלה 2)*

*אנו רוצים שתהיה קיימת ריצה, כך שבסוף יש מצב שבו אין אף משתנה מסוג cell שהוא עדיין goal.*

*CTL EF !(*

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)
2. 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-2)