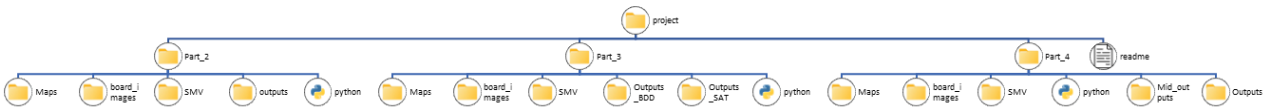
Git

repo: https://github.com/turbh11/sokoban_nuxmv.git

folder hierarchy:



PART 1

1. הגדרת הFDS:

$$\text{FDS}(sokoban) = \begin{cases} V: board\ (M*N) \cup shift \cup \{R\ able, L\ able, D\ able, U\ able\} \\ \theta: XSB\ map \\ \rho: non-detrmistic\{\rho_{up}, \rho_{down}, \rho_{left}, \rho_{right}\} \\ J: BIC \mid BNW \mid TBNW \\ C:\ \emptyset \end{cases}$$

משתנים:

board הינו מערך כל כלל המשבצות בלוח

Shift - מבטא את התזוזה הבא

4 – R able,L able,D able,U able מערכים שמבטאים האם 'חוקי' לבצע מהלך בכיוון זה בכל נקודה.

המערכים מוגדרים על פי הנוסחא הבאה:

$U\ able1[x][y] = (board[x-1][y] == \ floor \vee \ goal)$;
$U\ able2[x][y] = (x > \ 1)\ \&\ (board[x-1][y] == \ box \vee \ box\ on\ goal)\ \&\ \ (board[x-2][y] == floor \vee \ goal)$
$U\ able\ [x][y] = U\ able2[x][y] \vee U\ able1[x][y]$

(נעשה כאן פיתוח רק לתזוזה כלפי מעלה, ניתן להרחיב לשאר הכיוונים באופן זהה.)

תנאי ההתחלה:

תנאי ההתחלה מוגדרים להיות על פי מפת הXSB שניתנת למשחק.

פונקציית המעבר:

בכל "תזוזה" יש 4 מצבים אפשריים לכל משבצת בלוח:

i. השחקן נמצא במקום זה ועובר למשבצת ליד
ii. השחקן נמצא במשבצת ליד ועובר למשבצת זו
iii. השחקן נמצא 2 משבצות ליד ודוחף קובייה למשבצת זו
iv. השחקן זז בין משבצות אחרות ומשבצת זו ללא השפעה.

נגדיר את פונקציית המעבר לכל משבצת עבור 3 המצבים בהם יש שינוי במשבצת, יש לשים לב כי קיים שוני בין משבצות המוגדרות כמטרות , לבין משבצות המוגדרות כרצפה:

$\rho_{up1}(V[x][y]): U\ able[x+1][y] \wedge next\ (shift) == \text{U} \wedge (board[x+1][y] == \text{ warehouse keeper} \vee \text{ warehouse keeper on goal}) \wedge (board[x][y] == floor) \ \rightarrow board[x][y] = \text{warehouse keeper}$
$\rho_{up2}(V[x][y]): U\ able[x+1][y] \wedge next\ (shift) == \text{U} \wedge (board[x+1][y] == \text{ warehouse keeper} \vee \text{ warehouse keeper on goal}) \wedge (board[x][y] == goal) \ \rightarrow board[x][y] = \text{warehouse keeper on goal}$
$\rho_{up3}(V[x][y]): U\ able[x][y] \wedge next\ (shift) \ == \ U \wedge (board[x][y] \ == \text{warehouse keeper on goal} \rightarrow board[x][y] = goal$
$\rho_{up4}(V[x][y]): U\ able[x][y] \wedge next\ (shift) \ == \ U \wedge (board[x][y] == \text{warehouse keeper}) \rightarrow board[x][y] = floor$
$\rho_{up5}(V[x][y]): U\ able[x+2][y] \wedge next\ (shift) == \ U \wedge (board[x+2][y] == \text{warehouse keeper} \vee \text{ warehouse keeper on goal}) \wedge (board[x+1][y] \ == box \vee \ box\ on\ goal) \wedge (board[x][y] == floor) \ \rightarrow board[x][y] = box$
$\rho_{up6}(V[x][y]): U\ able[x+2][y] \wedge next\ (shift) == \ U \wedge (board[x+2][y] == \text{warehouse keeper} \vee \text{ warehouse keeper on goal}) \wedge (board[x+1][y] \ == box \vee \ box\ on\ goal) \wedge (board[x][y] == goal) \ \rightarrow board[x][y] = box\ on\ goal$

מכיוון שכל התנאים בתזוזות שהוצגו כאן סותרים אחד את השני ולא יכולים להתקיים במקביל, ניתן לבצע איחוד בין פונקציות המעבר ללא הגבלה.

$$\rho_{up} = \rho_{up1} \cup \rho_{up2} \cup \rho_{up3} \cup \rho_{up4} \cup \ \rho_{up5} \cup \rho_{up6}$$

(נעשה כאן פיתוח רק לתזוזה כלפי מעלה, ניתן להרחיב לשאר הכיוונים באופן זהה.)

יש לציין – במידה והשחקן זז בכל אחד מהכיוונים, ומשבצת זו הינה ללא השפעה, היא נשארת זהה למצבה הקודם.

JUSTICE:

את דרישת הjustice הגדרנו בשלושה חלקים שכל אחד מהם נדרש להתקיים (ולכן יש פעולת OR בין כל החלקים):

א. BIC – box_in_corner
הדרישה הזאת מבטיחה שלא תהיה קוביה בכל אחת מהפינות בלוח למשך אינסוף מהלכים, מכיוון שלא ניתן להוציא קוביה מפינה, הדרישה מבטיחה הימנעות מפינות.

ב. BNW – box_near_wall
הדרישה מבטיחה שלא תהיה קוביה אינסוף פעמים בכל אחת מהמשבצות שסמוכות לקיר בין שתי פינות, מכיוון שבמקרה כזה לא ניתן להרחיק את הקוביה מהקיר, הדבר יבטיח הימנעות מדחיפת קוביה לקירות שלא ניתן להיחלץ מהם (אלא אם כן קיימת שם מטרה)

ג. TBNW – two_box_near_wall
הדרישה מבטיחה שבכל שתי נקודות בלוח שסמוכות לקיר באחד מהמצדדים שלהם (לאו דווקא בין פינות), לא יהיה שתי קוביות צמודות למשך אינסוף מהלכים, מכיוון שבמקרה כזה אין אפשרות להזיז את הקוביות הדבר מונע היכנסות לdeadlock של 2 קוביות צמודות שלא ניתן להזיז.

(לצורך הגדרת הjustice הרצויים כדי להימנע מdeadlocks במשחק - השתמשנו במאמר זה)

2. הגדרת תנאי לניצחון:

בשביל ניצחון צריך שכל הקוביות יהיה על גבי מטרות
כלומר שבכל מקום בו יש מטרה או קופסה על מטרה במפת הXSB יהיה בסוף קופסה על מטרה.
התנאי המוגדר הוא eventually.

$$F\big((board[i_0][j_0] = box\ on\ goal) \wedge (board[i_1][j_1] = box\ on\ goal) \wedge \cdots \wedge (board[i_n][j_n] = box\ on\ goal)\big)$$

כאשר i,j הינם מערכים המבטאים את מיקומי המטרות (goal ∨ $box\ on\ goal$)
בקובץ הXSB (כלומר – בתנאי ההתחלה שלנו)

תעדוף: (רלוונטי לחלק 4.)

בנוסף, על מנת למנוע תקיעה במקרה של הרבה מטרות שנמצאות בסמיכות, בכך שנציב קוביות במטרות החיצוניות ונחסום גישה למטרות הפנימיות, הוספנו תעדוף לכל מטרה, מטרה רגילה הינה בעלת תעדוף '1' (נמוך) ומטרה פנימית הינה בעלת תעדוף גבוה יותר (התעדוף עולה ככל שהמטרה פנימית יותר).

בחנו מספר שיטות תעדוף ולבסוף בחרנו בשיטה זו שנותנת לדעתנו ציונים הכי ריאליים בהתאם למגבלות קיימות (קירות, מטרות שכנות, גמישות לדחוף קוביה לשם וכו')

התעדוף הבסיסי של כל מטרה נקבע על פי 2 פרמטרים (כאשר תעדוף הבסיס הוא 1):

א. שכנים-
עבור על אחד מהכיוונים (למעלה, למטה, שמאלה, ימינה) נבדק מה מוצב שם:
- קיר – התעדוף מקבל+2
- מטרה – התעדוף מקבל+1
- רצפה – התעדוף מקבל -1
- 2 רצפות ברצף – התעדוף מקבל -2

(נעשתה התעלמות מקוביות/הבן אדם בשל העובדה שהם דברים בעלי תזוזה והם נחשבו כמשבצת שעליהם הם עומדים רצפה/מטרה)

ב. תעדוף –
במקרה של רצף מטרות ניתן תעדוף לכל אחת מהמטרות על פי ההנחות הבאות:
- במידה ויש קיר צמוד באחד הצדדים – המטרה הפנימית מקבל את התעדוף הגבוה ביותר, ואז ברצף יורד עד לחיצונית שמקבלת תעדוף 1.
- במידה ואין קיר צמוד – המטרה האמצעית מקבל את התעדוף הגבוה ביותר בתצורת פעמון, כאשר החיצונית את התעדוף הנמוך ביותר - 1.

הדבר מבוצע במאונך (משמאל לימין ולהיפך) ובמאוזן (מלמעלה למטה ולהיפך), וכל קוביה מתקבלת תעדוף על סמך סכום התעדופים המקסימליים שנקבעו לה בכל כיוון (מאונך ומאוזן)

לאחר חיבור 2 השיטות במידה וקיימת מטרה בעלת תעדוף קטן מ1 , היא מוצבת לערך הדיפולטי (כלומר 1).

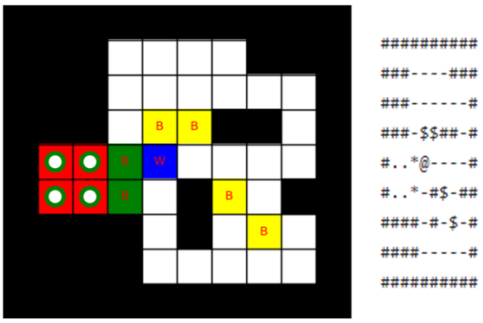
PART 2

.1

הקובץ נמצא בתיקיית הGIT תחת תיקיית "part_2", תידרש התאמת הPATH המוגדרים בתחילת הקובץ.

.2+3

כלל המפות שהורצו נמצאות בתיקיית MAPS
קיימות תמונות של המפות בתיקיית board_images

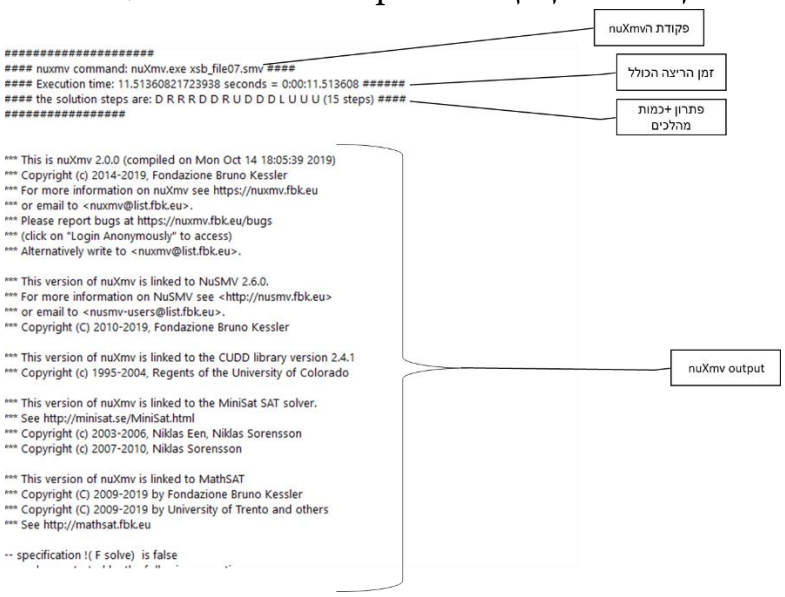


איור 1:מפה 8 בפרומט XSB ותמונה שנוצרת ע"י התוכנה

כלל התוצרים נמצאים בתיקיית OUTPUTS
כל קובץ מכיל הערה בתחילתו המכילה את :

א. פקודת ההרצה שבוצעה עבורו
ב. זמן הריצה שלקח לnuxmv להגיע לפתרון
ג. הפתרון (במידה ויש) בפורמט LURD

לאחר מכן מכיל הקובץ את הoutput של הnuxmv.(במידה ויש)



איור 2:מבנה קובץ OUTPUT של התוכנה

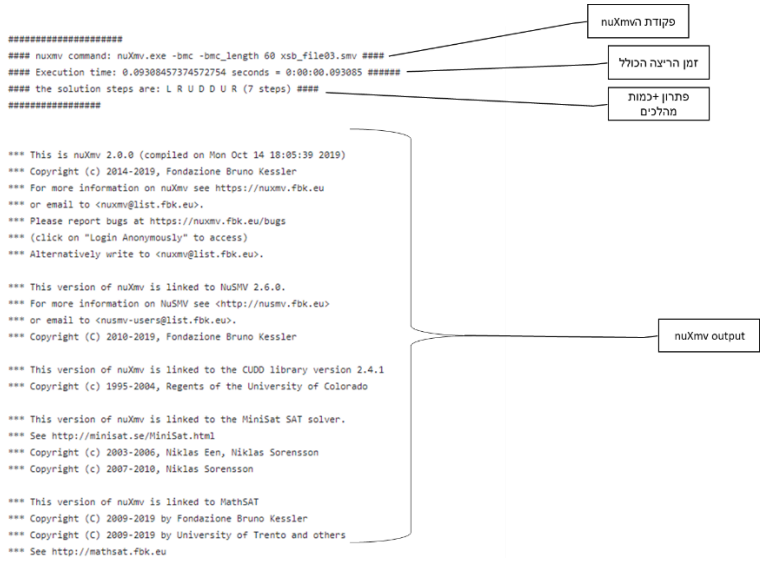
PART 3

.1

הקובץ נמצא בתיקיית הGIT תחת תיקיית "part_3", תידרש התאמת הPATH המוגדרים בתחילת הקובץ.
כלל המפות שהורצו נמצאות בתיקיית MAPS
קיימות תמונות של המפות בתיקיית board_images
כלל התוצרי הBDD נמצאים בתיקיית OUTPUTS_BDD
כלל התוצרי הSAT נמצאים בתיקיית OUTPUTS_SAT
כל קובץ מכיל הערה בתחילתו המכילה את :

א. פקודת ההרצה שבוצעה עבורו
ב. זמן הריצה שלקח לnuxmv להגיע לפתרון
ג. הפתרון (במידה ויש) בפורמט LURD

לאחר מכן מכיל הקובץ את הoutput של הnuxmv.(במידה ויש)



איור 3:מבנה קובץ OUTPUT של התוכנה

.2

מפה	מהלכים	זמן BDD (sec)	זמן SAT (sec)
0	1	0.030157	0.0202
3	7	0.073950	0.0930
6	11	66.7537	0.801
7	15	11.4940	4.089
8	-	Time-limit	Time-limit
9	23	2.02895	17.409
10	25	15.9472	423.3455
11	-	Time-limit	Time-limit

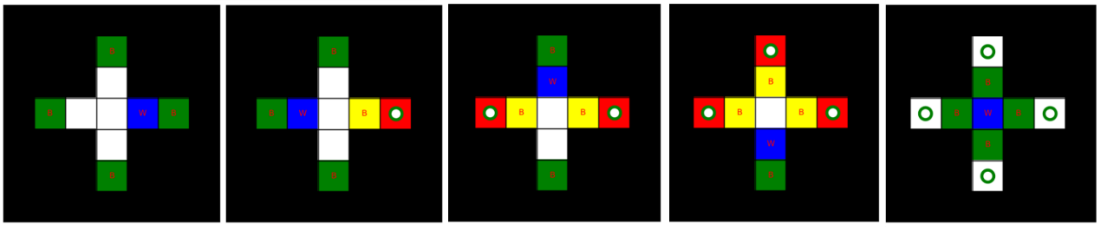
טבלה 1 : זמני ריצה בהרצת BDD וSAT על כלל המפות הפתירות שנבדקו

ניתן לראות כי אין בדיוק הצלחה חד חד ערכית של מנוע מסויים כזה או אחר....
מנוע הBDD מצליח להביא תוצאות טובות יותר בלוחות בעלי מספר מהלכים רב (+20) ומנוע הSAT בעל תוצאות יותר טובות בלוחות עם מספר מהלכים קטן יותר (~10).

PART 4

התנאי שהוגדר על ידינו הינו שבסוף האיטרציה הנוכחית תהיה קוביה אחת יותר על מטרה מאשר במצב בו התחלנו, כלומר אם בתחילת האיטרציה ישנה קוביה אחת שכבר נמצאת על מטרה, בסוף נדרשות להיות 2 קוביות שנמצאות על מטרות – ללא תלות האם אחת מהן היא הקוביה איתה התחלנו את האיטרציה.
(העלאת קוביה למטרה בעלת "תעדוף" גבוה יותר נחשבה אצלנו כהצבת קוביה נוספת וגרמה גם לסיום איטרציה- ולכן קיימים לוחות עם יותר איטרציות מאשר מספר הקוביות)

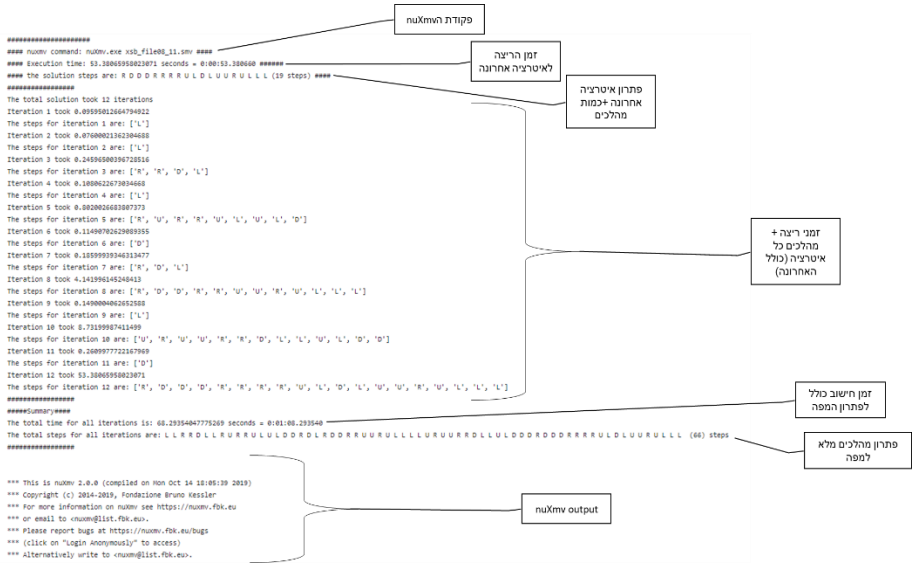
הקובץ נמצא בתיקיית הGIT תחת תיקיית "part_4", תידרש התאמת הPATH המוגדרים בתחילת הקובץ.
כלל המפות שהורצו נמצאות בתיקיית MAPS
כלל התוצרי ההרצה נמצאים בתיקיית mid_OUTPUTS
כאשר n_ בסוף כל קובץ מבטא את מספר האיטרציה.
קיימות תמונות של המפות **(לכל איטרציה)** בתיקיית board_images



איור 4:מידול מפה 3 ע"י התוכנה במהלך 4 האיטרציות שנדרשו לפתרונה + תמונת פתרון סופי.

כלל התוצרים הסופיים נמצאים בתיקיית OUTPUTS
כל קובץ מכיל הערה בתחילתו המכילה את :

א. כמות האיטרציות שנלקחו לפתרון.
ב. זמן הריצה שלקח לnuxmv להגיע לפתרון כל איטרציה + זמן כולל.
ג. פירוט המהלכים לניצחון בכל איטרציה + באופן כולל.



איור 5:מבנה קובץ OUTPUT של התוכנה בהרצת איטרציות

מפה	מהלכים	זמן BDD (sec)	זמן SAT (sec)	איטרציות :	
				מהלכים	זמן (sec)
3	7	0.073950	0.0930	7	0.2369
6	11	66.7537	0.801	11	1.0899
7	15	11.4940	4.089	15	0.8189940
8	-	Time-limit	Time-limit	66	76.9691
9	23	2.02895	17.409	23	11.48967
10	25	15.9472	423.3455	25	3.4290
11	-	Time-limit	Time-limit	252	8324.4296

טבלה 2 : זמני ריצה בהרצת SAT ו BDD על כלל המפות הפתירות שנבדקו אל מול הרצת איטרציות

ניתן לראות כי לשיטה זו יתרון מובהק על פני ביצוע של כל הלוח בפעם אחת, בחינה של מפה 8 לצורך הדוגמא מראה כי בהרצה של כלל הלוח בפעם אחת לא הצלחנו להגיע לפיתרון בתוך שעתיים (גם בBDD ובSAT) בעוד שבהרצת איטרציות קיבלנו את הפתרון של מפה 8 בתוך כ-דקה ורבע.

הדבר נובע מכך שהסיבוכיות של החישוב הינה 4^n ולכן ככל שמספר המהלכים גדל הסיבוכיות עולה בהרבה, ופיצול שומר על מספר מועט של מהלכים ומאפשר להקטין את הסיבוכיות (כלומר- עדיף 8 פעמים פתרון של 8 מהלכים מאשר פתרון של 64 במכה)