האוניברסיטה הפתוחה

המחלקה למתמטיקה ולמדעי המחשב

המרת מודל בשפת Esterel בצורה אוטומטית לקלט עבור הכלי NuSMV, לצורך אימות המודל, זאת תחת הנחת קיום סיגנלים טהורים בלבד

Final project by Tehila Hanunov

פרויקט מתקדם לתואר שני - תהילה חנונוב

Guided by Prof. Shmuel Tyszberowicz

בהדרכתו של פרופ׳ שמואל טישברוביץ

נובמבר 2020

TABLE OF CONTENTS

מטרת הפרויקט 3

רקע 3

הגישה הסינכרונית למערכות זמן אמת 5

הצורך במודל הסינכרוני 5

עקרונות הגישה והמודל המתמטי עליו היא מסתמכת 5

Esterel 7

Model Checking 10

המודל של Kripke 10

NuSMV 11

אלגוריתם ההמרה 12

הגדרת משתני הקלט 12

הגדרת משתנה העזר await 13

הגדרת משתני הפלט 13

הגדרת השעון הגלובלי ב- NuSMV 14

התמודדות עם לולאות 14

הגדרת next עבור כל סיגנל פלט 15

התמודדות עם emit 15

התמודדת עם present 16

התמודדת עם sequence 17

התמודדות עם מקביליות 17

התמודדות עם Pause 18

דוגמת המרה – ABRO 18

אימות באמצעות NuSMV 21

הבטחת FAIRNESS 21

המרת תכונות לבדיקה 21

דוגמא א' – ABRO 21

דוגמא ב' 22

קוד הפרויקט 24

רשימת מקורות 39

מטרת הפרויקט

מטרת הפרויקט הינה בניית מודל אימות לשפת Esterel על ידי שימוש ב NuSMV תחת הנחת קיום סיגנלים טהורים בלבד (ללא מספרים שלמים). כחלק מעבודתי המסכמת חקרתי את הגישה הסינכרונית למערכות תגובתיות ומערכות זמן אמת. ישנן מספר שפות אשר מממשות את הגישה ו- Esterel ביניהן. בנוסף, ישנם גם כלים אשר פותחו לאימות שפות סינכרוניות אך השימוש בהן מעורר קשיים ואינו מספק חוויה מספיק טובה למשתמשים. מנגד, NuSMV, הוא כלי לאימות מכונת מצבים סופית והוא הוכיח עצמו בשנים האחרונות כשימושי ביותר.

מטרת הפרויקט היא לשלב בין השניים ולהשתמש ביכולותיו של NuSMV על מנת לאמת את נכונותן של תכניות Esterel. זאת על ידי המרת תכניות Esterel לקוד של NuSMV ובדיקת נכונותו. ההנחה כי במודל ה-Esterel קיימים סיגנלים טהורים בלבד נובעת מכך שאם נאפשר את קיומם של מספרים שלמים לא נוכל לבצע אימות עקב בעיית התפוצצות המצבים.

רקע

הגישה הסינכרונית למערכות תגובתיות ומערכות זמן אמת החלה להתפתח בשלהי סוף ה-80. המונח מערכת תגובתית [4] הוטבע כדי לנסות לתת הגדרה מדויקת למערכת מסוג זה והוא קובע כי מערכת היא תגובתית במידה והיא שומרת על אינטראקציה קבועה עם הסביבה שלה ובנוסף קצב עבודתה נקבע ע"י הסביבה. כלומר, כאשר היא מקבלת קלט חיצוני מהסביבה (ע"י סנסור או אירוע מסוים) ומוציאה כפלט פקודות בחזרה לסביבה. מערכת זמן אמת היא כזו שבה כל תגובה חייבת להסתיים בצורה מידית. בניית מתודולוגיות וגישות יעילות לפיתוח מערכות זמן אמת הן חיוניות כיוון שהשכיחות של מערכות אלו בתעשייה ובחיי היום יום שלנו הולך וגדל, והוא בא לידי ביטוי כמעט בכל תחום סביבנו - במפעלים, מטוסים, מכוניות, פלאפונים, טלוויזיות חכמות ועוד הרבה.

החידוש בגישה הסינכרונית היה ההנחה כי המערכות בהן דנים הן סינכרוניות וכי הן יודעות לחשב את הפלט בצורה מידית ברגע שהן מקבלות את הקלט ובנוסף הן מאפשרות להתמודד עם פעילויות מקביליות . הנחה זו הופכת את החיים ל"קלים יותר" ומאפשרת לתאר ולנתח מערכות אלו בצורה נוחה הרבה יותר. הרעיון העומד מאחורי הנחה זו הוא כי ניתן "לעצור" את הסביבה ולטפל באירועים שקרו בזמן התגובה הנוכחית רק בתגובה הבאה בתור וכך נשמר עקרון הדטרמיניזם. הנחת הסינכרוניות מאפשרת לחלק את המערכת לתתי-רכיבים הרצים במקביל מבלי שתהיה השפעה כלפי חוץ.

Esterel [4] היא אחת השפות אשר מממשות גישה זו המשלבת את עקרונות הסינכרוניות והמקביליות הדטרמיניסטית. שפת Esterel היא אימפרטיבית (מורכבת מפקודות) ומתאימה לתיאור מערכות שליטה ובקרה.

עבור אימות פורמלי של מערכת כלשהי נדרשים: (1) מודל מתמטי של המערכת (2) שפה לקביעת המאפיינים הרצויים של המערכת בצורה תמציתית, מובנת וחד משמעית ו-(3) שיטת הוכחה לאימות המאפיינים שצוינו. ***בדיקת מודל סמלית*** *(Symbolic Model Checking)* [1] עוסקת בשיטות של אימות אוטומטי (על ידי מכונה) על גבי חומרת מחשב.

המוטיבציה לפתח תחום זה הייתה העלות הגבוהה והגוברת של תיקון שגיאות תכנון לאחר תחילת הייצור. מתודולוגית עיצוב שתגלה שגיאות עוד לפני תחילת הייצור, ואפילו עוד בזמן

העיצוב, יכולה לחסוך כסף, מאמץ הנדסי ואי עמידה בלוחות זמני הפיתוח. מלבד שיקולי עלות טהורה, ישנו גם צורך באספקט התיאורטי והוא לספק בסיס מתמטי חזק לתכנון של מערכות מחשב.

בכדי ליישם את בדיקת המודל הסמלית בבעיות מהעולם האמיתי עלה צורך בשפה שתאפשר לתאר את המודל בשפה גבוהה (high level). לשם כך נועדה *SMV* [1] . SMV היא כלי לבדיקת מערכת מצבים סופית אל מול מפרט (ספציפיקציה) הנכתב ב-*CTL (Computation Tree Logic)* [2].

NuSMV [3] היא הרחבה ומימוש מחודש של SMV. זהו כלי לבדיקת נכונותו של מודל המבוסס על דיאגרמות החלטה בינאריות BDD (Binary Decision Diagrams), אשר נוצר כתוצאה משיתוף פעולה בין המוסד FBK-IRST (Bruno Kessler Center in Information and Communication Technology) ואוניברסיטת Carnegie Mellon. הכלי עוצב בארכיטקטורה פתוחה (open architecture) המאפשרת הוספה, שדרוג והחלפה של רכיבים בצורה קלה. פיתוחו של הכלי נועד בכדי לשרת מספר מטרות מרכזיות. ראשית כל, בכדי לספק כלי אמין לאימות של מערכות אמתיות מהתעשייה, בגדלים אמיתיים. שנית, הכלי משמש כצד השרת עבור כלים אחרים לאימות המרחיבים את יכולותיו של כלי זה. שלישית, זהו כלי מחקרי לשיטות אימות פורמליות.

הגישה הקלאסית של אימות ובדיקות במודל הסינכרוני עושה שימוש במשקיפים (Observers) [4]. משקיפים סינכרוניים הם כלי לאפיון תכונות של תכניות ולתיאור התנהגויות לא דטרמיניסטיות. אימות ע"י משקיפים היא דרך מאוד טבעית בשפות סינכרוניות. תכונות בטיחות יכולות להיות מוגדרות ע"י תכנית מיוחדת הנקראת משקיף אשר משקיפה על משתנה או סיגנלים מעניינים ובכל צעד קובעת האם התכונה מתקיימת עד לצעד זה ע"י שליחת סיגנל המסמל זאת. תכנית עומדת בתכונת הבטיחות אם ורק אם המשקיף לעולם לא מופר באף הרצה. ישנם מספר כלים ובהן XEVE [5] שמשתמשים בטכניקה זו. אך, כלים אלו אינם מוכחים עצמם כאינטואיטיביים ואמינים מספיק. זאת בעיקר משום שהגרסאות העדכניות של כלים אלו אינם נתמכים יותר באקדמיה.

מטרת הפרויקט היא לשלב בין השניים ולהשתמש ביכולותיו של NuSMV על מנת לאמת את נכונותן של תכניות Esterel. זאת על ידי המרת תכניות Esterel לקוד של NuSMV ובדיקת נכונותו. ההנחה כי במודל ה-Esterel קיימים סיגנלים בלבד נובעת מכך שאם נאפשר את קיומם של מספרים שלמים לא נוכל לבצע אימות עקב בעיית התפוצצות המצבים.

הגישה הסינכרונית למערכות זמן אמת

הצורך במודל הסינכרוני

במערכות זמן אמת קריטיות קיים צורך עז בהוכחת נכונות העיצוב והמימוש על מנת להבטיח את ביטחון המשתמשים והסביבה. בעבר, לא היו קיימות מתודולוגיות וכלים אשר אפשרו תכנון של מערכות מסוג זה בצורה שתשלב בין דטרמיניזם ומקביליות (מתכנן המערכת נאלץ לבחור באחד מן השניים בלבד) ולכן עלה הצורך להשקיע בפיתוח מתודולוגיות מתאימות. מתודולוגיות אלו היו צריכות לענות על שני קריטריונים עיקריים. האחד - היה עליהן להשתלב בצורה חלקה עם צורת העבודה והכלים הקיימים. הסיבה לכך היא ברורה, כל כלי חדש אשר אינו נמצא כבר בשימוש על ידי המתכנן יוריד את הביטחון שלו בעיצוב, דבר שלא ניתן לאפשר במערכות מסוג זה. הקריטריון השני היה בניית המתודולוגיות והכלים בהתבסס על עקרונות מתמטיים מוצקים כך שיתאפשר אימות פורמלי של נכונות ותפקוד המערכת. תכונה זו תאפשר בנוסף לייעל את המימוש ע"י תמיכה בדרישות לא פונקציונאליות.

הבנות אלו הובילו למסקנה שיש צורך בהגדרת שפות סינכרוניות שעליהן לממש את העקרונות הבאים:

1. מקביליות - שפות אלו חייבות לתמוך במקביליות וביכולת להגדיר ארכיטקטורה למערכת הבנויה מרכיבים הפועלים במקביל. על מנת לעשות זאת יש להגדיר תחביר מקבילי ידידותי למשתמש. תחביר זה צריך להתבסס על תחביר מוכר (כגון דיאגרמת בלוק -block diagram, או אוטומט היררכי - automata hierarchical) בקהילת המתכנתים בכדי לאפשר הטמעה מהירה וקלה.
2. סינכרוניות - על שפות אלו להיות מסוגלות לחשב את הפלט מידית עם קבלת הקלט (זאת בהנחה שכל פעולה לוקחת זמן וזיכרון סופי).
3. פשטות - המודל צריך להיות פשוט ככל האפשר על מנת שיהיה ניתן לעקוב אחריו ולהבין אותו בקלות. עקרון זה חשוב במיוחד עבור הייצוג המקבילי שצריך להיות נקי ככל האפשר.

בנוסף להצגת הגישה הסינכרונית הוצגו גם השפות Esterel, Lustre ו- Signal אשר בנויות על מערכת מתמטית המשלבת סינכרוניות ומקביליות דטרמיניסטית. בזמנו שלוש השפות היו מוגדרות היטב ונעשה בהם מעט שימוש מסחרי אך הן היו עדיין תחת פיתוח. עם השנים, הוכנסו לשפות שיפורים שונים, קהילת המשתמשים התרחבה משמעותית והשימוש בתעשייה גדל גם כן. כיום, משמשות השפות הסינכרוניות כטכנולוגיה מרכזית למידול, ספציפיקציה, ולידציה ומימוש של מערכות זמן אמת.

עקרונות הגישה והמודל המתמטי עליו היא מסתמכת

השפות הסינכרוניות הצליחו לשלב מקביליות, סינכרוניות ופשטות למרות שזו אינה משימה קלה. סינכרוניות משמעותה חלוקת הזמן למופעים דיסקרטיים, כפי שנהוג לעיתים במתמטיקה והנדסה. לכן, היה זה טבעי להחליט שתוכניות סינכרוניות יתקדמו בהתאם **לתגובות** אטומיות (*atomic reactions*) רציפות. מכאן ואילך נתייחס לתגובות/צעדים כמילים נרדפות.

המשוואה המבטאת עקרון זה נכתבת בצורה הבאה: $P \equiv R^w$. כאשר:
P היא תכנית סינכרונית
R מייצג את אוסף כל התגובות האפשריות
w מייצג את מופעי הזמן האינסופיים (האינטרוולים).

בדיאגרמת בלוק של מערכות שליטה ובקרה, התגובה בזמן (אינטרוול) ה-n של המערכת, היא שילוב של n התגובות של כל אחד מהרכיבים הפועלים בה.

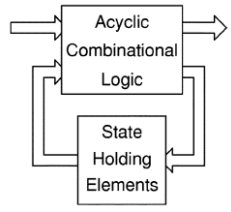
$$\mathrm{Y}_n^i = g(\mathrm{X}_{n-1}^i, \mathrm{U}_n^i) \quad ; \quad \mathrm{X}_n^i = f(\mathrm{X}_{n-1}^i, \mathrm{U}_n^i) \qquad (1)$$

כאשר U הוא וקטור הקלט, X הוא וקטור המצב ו-Y הוא וקטור הפלט. קומבינציה משמעותה שקלט או פלט כלשהו של בלוק i קשור בצורה כלשהי לקלט של בלוק j למשל:

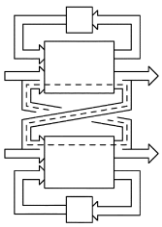
$$\mathrm{Y}_n^i(k) = \mathrm{U}_n^i(l) or\ \mathrm{Y}_n^i(l) \quad (2)$$

כאשר $\mathrm{Y}_n^i(k)$ מייצג את הקוארדינטה ה-*k* של וקטור הפלט של בלוק *j* בזמן *n*. כלומר, תגובה של מערכת המורכבת ממספר בלוקים שונים היא איחוד התגובות של כל בלוק (1) כולל הקשרים בין הבלוקים. למשל התגובה בתרשים *1(b)* מייצגת שני בלוקים כאשר התגובה של כל אחד מהבלוקים היא חלק מתגובה במערכת אליה הם משתייכים.

תרשים 1(b)



תרשים 1(a)



תרשים 1(a) מראה כיצד מערכת מצבים סופית (*finite state machine (FSMs)*) ממומשת לרוב בחומרה: בלוק חסר מעגלים מחשב את הפלט ואת המצב (state) הבא כפונקציה של הקלט ושל המצב הנוכחי.

תרשים 1(b) מציג את הדרך הטבעית ביותר להריץ שני אוטומטיים סופיים (FSM) במקביל ולאפשר להם לתקשר ביניהם, זאת על ידי חיבור חלק מהפלט של אוטומט אחד כקלט לאוטומט השני ולהפך. הקו המקווקו מייצג נתיב לפידבק מידי כתוצאה מחיבור הפונקציונליות של שני האוטומטים.

כתוצאה מכך, הצורה הטבעית שנבחרה לייצוג של מקביליות שני רכיבים P_1 ו-P_2 היא $P_1 \parallel P_2 \equiv (R_1 \wedge R_2)^w$. כאשר כל תכנית מוגדרת בדומה למשוואה 3 (כפי שראינו קודם): R_1 ו-R_2 הם אוסף התגובות האפשריות ו-w מייצג את מופעי הזמן הבדידים האינסופיים.

זאת אומרת שמודל הסינכרוניות יכול להיות מסוכם ע"י שתי המשוואות הבאות:

$$P \equiv R^w \qquad (3)$$

$$P_1 \parallel P_2 \equiv (R_1 \wedge R_2)^w \qquad (4)$$

הגדרת משוואה 4 דורשת למעשה את הגדרת החיתוך $(R_1 \wedge R_2)$ של שתי תגובות של שני רכיבים שונים. בדרך כלל האיחוד יוגדר ע"י יחס בין שני רכיבים או אילוץ. למשל, דוגמא להגדרת איחוד תגובות באמצעות אילוץ: שני אוטומטים חייבים להסתנכרן כאשר מתבצעת טרנזקציה משותפת.

טבלה 1

emit S	Make signal S present immedi-ately
present S then p else q end	If signal S is present, perform p otherwise q
pause	Stop this thread of control until the next reaction
p ; q	Run p then q
loop p end	Run p; restart when it terminates
await S	Pause until the next reaction in which S is present
p ‖ q	Start p and q together; terminate when both have terminated
abort p when S	Run p up to, but not including, a reaction in which S is present
suspend p when S	Run p except when S is present
sustain S	Means loop emit S; pause end
run M	Expands to code for module M

שפת Esterel היא אימפרטיבית (מורכבת מפקודות) ומתאימה לתיאור שליטה ובקרה. זאת בניגוד לשפת Lustre אשר הינה שפה הצהרתית (דקלרטיבית) המתמקדת בהגדרת זרימת המידע. באופן אינטואיטיבי, תכנית Esterel מורכבת מאוסף של תהליכים (threads) מקוננים הרצים במקביל. התחביר הוא אימפרטיבי מסורתי, כאשר הביצוע מסונכרן לשעון אחד גלובלי. בטבלה 1 ניתן לראות מספר פקודות בסיסיות בשפה.

בתחילת כל צעד/תגובה כל תהליך (thread) ממשיך את הביצוע שלו מהנקודה האחרונה בה נעצר בתגובה הקודמת, הוא מבצע את הפקודות ולבסוף מסיים או עוצר כהכנה לתגובה הבאה.

חוטים מתקשרים אך ורק באמצעות סיגנלים, סיגנל הוא הטיפוס הראשי ב- Esterel אשר מייצג אירוע של שידור גלובלי.

כלל הקוהרנטיות הבא של השפה מבטיח שכלל תכניות Esterel יתנהגו בצורה דטרמיניסטית: כאשר thread בודק סיגנל מסוים במהלך תגובה הסיגנל תמיד יתקיים או לא יתקיים, אך לעולם הוא לא יראה את שניהם במהלך אותה התגובה. מטרת הסיגנל היא לייצג אירוע ולכן הוא אינו קבוע בין תגובות שונות. בצורה מדויקת יותר: סיגנל מתקיים בתגובה אם ורק אם התוכנית או הסביבה הגדירו כי הוא מתקיים, כלומר פלטו אותו.

הצהרות מקדימות (Preemption statements) אשר מאפשרות תיאור היררכי ונקי של התנהגות מכונת מצבים הן אחת החוזקות של השפה. בהצהרת if-then-else המסורתית הפרדיקטים נבדקים פעם אחת לפני הביצוע, ב-Esterel לעומת זאת הצהרה מקדימה (למשל abort) בודקת את הפרדיקטים בכל תגובה שבו גוף הלוגיקה רץ. חלק מההצהרות המקדימות מאפשרות לבחור האם לבדוק את הפרדיקט מידית/ לבדוק את הפרדיקט כאשר גוף הלוגיקה רץ / לבדוק את הפרדיקט לפני או אחרי ריצת הלוגיקה ועוד.

הסמנטיקה של Esterel מאפשרת מעגלים, על פניו, בניתוח הסטטי וייתכנו אפילו (מבוי סתום) deadlocks אך בפועל, בהרצה דינמית המעגל לעולם לא יופעל. הבדיקה כי בתוכנית Esterel אין מבוי סתום נקראת *causality analysis* והיא כוללת בדיקה שהאילוצים הסיבתיים לעולם אינם סותרים אחד לשני באף מצב ואף מתקיימת ביניהם תלות הדדית.

הסמנטיקה של Esterel מבוססת על constructive causality [9]. ההבנה של בעיה זו, והפתרון המתמטי הנקי שלה היא אחת מהתרומות המשמעותיות של Esterel ליסודות הסמנטיקה של מערכות תגובתיות סינכרוניות. חוסר הבנה מלאה של בעיה זו הוא מה שמנע מהמעצבים של השפות Verilog, VHDL להגיע לסמנטיקה סינכרונית מלאה. דבר שהצליחו להשיג ב-Esterel.

כבר מראשיתה, Esterel שמה את עקרונות האופטימיזציה, אנליזה וולידציה במרכז תהליך הקומפילציה. בנוסף, הוכנסה ל-Esterel טכניקה חדשה לקומפילציה/הידור: המהדרים הראשונים של השפה היו מבוססים על תרגום מילולי של הסמנטיקה של השפה ע"י שימוש בגישה של פלוטקין [10]. הגרסאות הבאות של הקומפיילרים (גרסאות 1 ו-2) התבססו על בניית אוטומט לכל תוכנית Esterel ע"י שימוש באלגוריתם של ברז'וזובסקי [11]. עבודה מאוחרת יותר של גונטיר [12] ואחרים עזרה לפתח קומפיילר אשר האיץ בצורה משמעותית את תהליך בניית האוטומט. שיטה זו התבססה על חיקוי המבנה של קוד ה-IC (קוד הביניים - intermediate code) - גרף זרימה מקבילי הנתלה על שחזור העץ המייצג את ההגדרות ההיררכיות המקוננות.

שיטת האוטומט (בגרסאות הראשונות של הקומפיילר) עבדה היטב עבור תכניות קטנות והיא ייצרה קוד מהיר ויעיל. אך, השיטה לא החזיקה מים בתוכניות גדולות המייצגות דוגמאות אמתיות בתעשייה כיוון שהיא נתקלה בבעיית "התפוצצות המצבים" (state explosion problem). מספר חוקרים ניסו לשפר את איכות הקוד של יצירת האוטומט על ידי מיזוג אלמנטים משותפים לצמצום הגודל ורעיונות דומים אך ללא הצלחה כבירה, אף אחד מהאופטימיזציות האלו לא הצליחו לקמפל תכניות בעלות יותר מ-1,000 שורות קוד .

הדור הבא של הקומפיילרים עבור Esterel התבסס על המרה ללוגיקה דיגיטלית (digital logic), המרה זו הייתה טבעית עקב סמנטיקת המצבים של Esterel. היתרון המרכזי של שיטה זו, בניגוד לשיטת ההמרה לאוטומט, הוא שההמרה היא כמעט "אחד לאחד" (כל שורת הגדרה במקור הופכת לכמה שערים לוגים לכל היותר) כך שהפתרון יכול להתרחב בקלות לתוכניות גדולות. אם כך, קבצי ההרצה הנוצרים מקומפיילרים בשיטה זו הם הרבה יותר קטנים אך עדיין ישנו מקום לשיפור. למשל, קיימות טכניקות להפחתת מספר ה- latches בקוד הנוצר ובכך לשפר את הגודל והמהירות של הקוד הנוצר.

הגרסה הראשונה של קומפיילר המבוסס על גישה זו (גרסה 4) נתקלה במתנגדים מצד המשתמשים כיוון שהוא לא התחשב בצורה נכונה בתוכניות שכבר עברו קומפילציה בגרסה הקודמת. הבעיה נבעה מכך שבגרסה החדשה התעקשו שהרשתות הדיגיטליים יהיו ללא מעגלים, כלומר תלויות המידע והזרימה צריכים להתקיים בכל מצב אפשרי, אפילו במצבים שאליהם התוכנית לעולם לא תגיע (בגרסאות הקודמות לעומת זאת התייחסו לכל מצב בנפרד ורק למצבים שאליהם התוכנית יכולה להגיע).

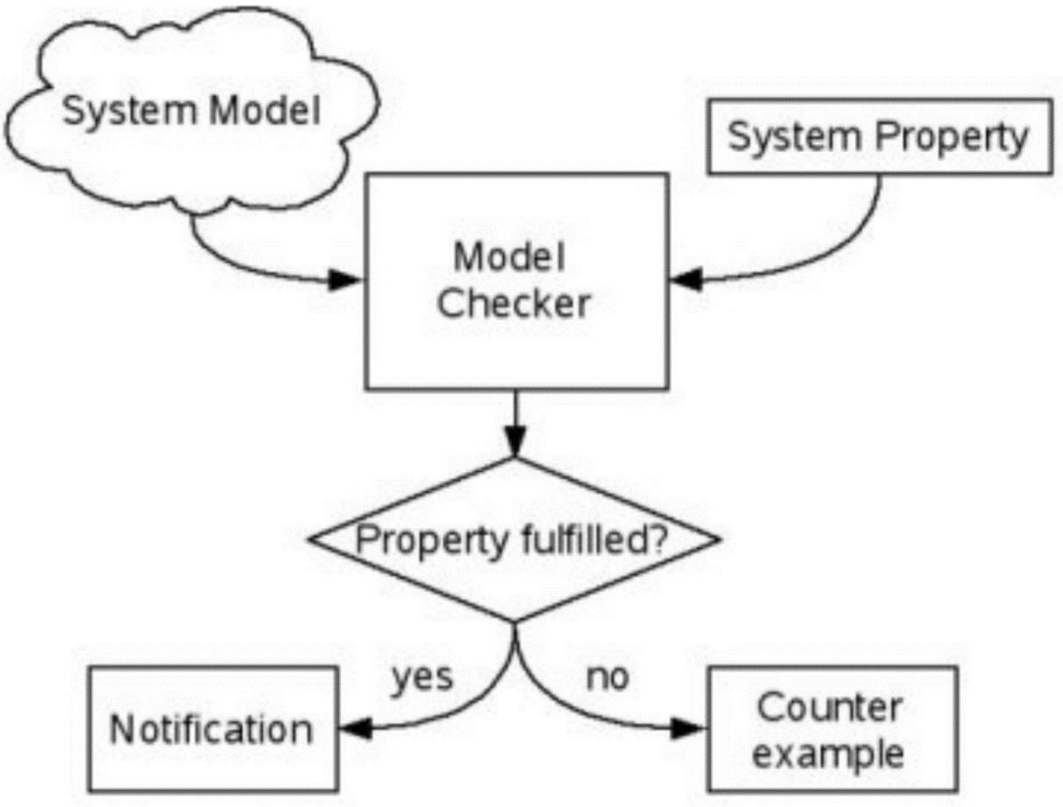
הפתרון למגבלה בעייתית זו היה בניה מחדש של הסמנטיקה של Esterel במסגרת מובנית. ההשראה נלקחה מעבודתו של מאליק [13] על מעגלים דיגיטליים קומבינטורים אשר אפשרה ב-Esterel את קיומם של מעגלים במצבים שאליהם התוכנית לא יכולה להגיע. החסרון הגדול של גרסת הקומפיילר שהתבססה על גישה זו (גרסה 5) היה איטיות יצירת הקוד (עד כדי פי 100 מקומפיילר מבוסס אוטומט) עקב זמן חישוב ארוך של המצבים שניתן להגיע אליהם.

שתי טכניקות חדשות (הקומפיילר של אדוארדס [14] והטכניקה של וייל ונוספים [15]) ניסו לשלב בין שתי הגישות. טכניקות אלו שתיהן מנסות לייצג תכנית Esterel כגרף בלוקים פשוט

המקושר ע"י זרימת מידע ותלויות ואז לתזמן בין הבלוקים ולייצר קוד בהתאם. גישות אלה ממומשות בקומפיילר SAXO-RT [16] ואומנם יש להם יתרונות אך הן רחוקות ממושלמות (עדיין ישנה מגבלת המעגלים בגישה של אדוארדס והגישה של וייל אינה מממשת את הסמנטיקה המלאה של השפה).

מטרתו של כלי Model Checker היא להוכיח האם מודל מסוים מקיים תכונה (property) כלשהי. בהינתן מודל M ונוסחה ϕ, ה- Model Checker יבדוק האם ϕ מתקיים ב- M.

תרשים 2



כפי שניתן לראות בתרשים 2. ה- Model Checker מקבל את תיאור המודל ואת התכונה אותה רוצים לבדוק. אם התכונה מתקיימת על המודל ה- Model Checker יחזיר תשובה חיובית. במידה והתכונה לא מתקיימת ה- Model Checker יחזיר דוגמא נגדית המסבירה מדוע התכונה אינה מתקיימת.

המודל של KRIPKE

את המודל, M, נהוג להגדיר באמצעות מערכת מצבים סופית. Kripke הגדיר את מבנה המודל באופן הבא: $M = (S, s_0, R, L)$ כאשר:

S - סט סופי של מצבים

s_0 - סט מצבים התחלתיים

R - יחס המעברים בין המצבים

L - פונקציית סימון המתאימה לכל מצב את הנוסחאות האטומיות שמתקיימות בו

את הנוסחה ϕ ניתן להגדיר באמצעות לוגיקה טמפורלית (CTL, LTL, CTL*)

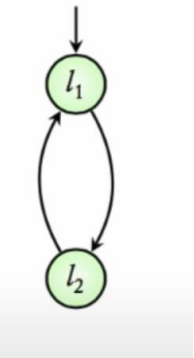
NUSMV

NuSMV [3] היא הרחבה ומימוש מחודש של SMV. הכלי פותח ב-1999, והוא מבוסס על מודל Kripke, מאפשר לתאר אוטומטים ולבדוק את נכונותו של המודל

זהו כלי לבדיקת נכונותו של מודל המבוסס על דיאגרמות החלטה בינאריות BDD (Binary Decision Diagrams), אשר נוצר כתוצאה משיתוף פעולה בין המוסד FBK-IRST (Bruno Kessler Center in Information and Communication Technology) ואוניברסיטת Carnegie Mellon. הכלי עוצב בארכיטקטורה פתוחה (open architecture) המאפשרת הוספה, שדרוג והחלפה של רכיבים בצורה קלה.

פיתוחו של הכלי נועד בכדי לשרת מספר מטרות מרכזיות. ראשית כל, בכדי לספק כלי אמין לאימות של מערכות אמתיות מהתעשייה, בגדלים אמיתיים. שנית, הכלי משמש כצד השרת עבור כלים אחרים לאימות המרחיבים את יכולותיו של כלי זה. שלישית, זהו כלי מחקרי לשיטות אימות פורמליות.

נביט בדוגמא למודל ב- NuSMV המאפיין מערכת של שני מצבים l1,l2 אשר מתבצע מעבר ביניהם בכל צעד.



קוד ה- NuSMV המתאים למערכת מצבים זו נראה כך:

```
MODULE main
VAR
    location:  {l1,l2};
ASSIGN
    init(location) := l1;
    next(location) := case
                (location = l1) :  l2;
                (location = l2) :  l1;
                     esac;
```

הקוד מגדיר כי ישנו משתנה אחד בשם location. למשתנה זה שני ערכים אפשריים l1,l2. הערך ההתחלתי של location הוא l1. הערך הבא (next) של location מוגדר כך: - אם הערך הנוכחי הוא l1 אז הערך החדש יהיה l2. - אם הערך הנוכחי הוא l1 אז הערך החדש יהיה l2.

אלגוריתם ההמרה

קלט: תוכנית בשפת Esterel

פלט: תוכנית מקבילה בשפת NuSMV

האלגוריתם בקווים כלליים בנוי מהשלבים הבאים:

- הגדרת משתני הקלט
- הגדרת משתני עזר עבור הקלט (await)
- הגדרת משתני הפלט ואתחולם
- הגדרת השעון הגלובלי
- זיהוי לולאות והתמודדות עמן
- זיהוי הצהרות מיוחדת כגון pause והתמודדות עמן
- הגדרת next עבור כל סיגנל פלט (באמצעות פרסור תוכנית ה- Esterel)

המימוש נעשה ב- python3.

כחלק מהמימוש נלקחו מספר הנחות בסיס:

- בקוד המקור ב- ESTEREL לא מופיעים מספרים שלמים
- כל הקוד של תוכנית ה- Esterel נמצא תחת קובץ אחד
- נוכחות של סיגנלי הפלט בזמן אפס תמיד תהיה FALSE
- ישנם מילים שמורות ב- NuSMV שיש להתעלם מהם:

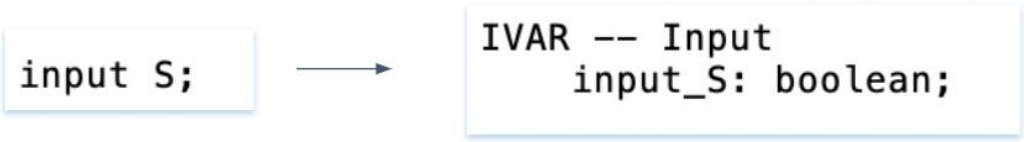
MODULE, DEFINE, MDEFINE, CONSTANTS, VAR, IVAR, FROZENVAR, INIT, TRANS, INVAR, SPEC, CTLSPEC, LTLSPEC, PSLSPEC, COMPUTE, NAME, INVARSPEC, FAIRNESS, JUSTICE, COMPASSION, ISA, ASSIGN, CONSTRAINT, SIMPWFF, CTLWFF, LTLWFF, PSLWFF, COMPWFF, IN, MIN, MAX, MIRROR, PRED, PREDICATES, process, array, of, boolean, integer, real, word, word1, bool, signed, unsigned, extend, resize, sizeof, uwconst, swconst, EX, AX, EF, AF, EG, AG, E, F, O, G, H, X, Y, Z, A, U, S, V, T, BU, EBF, ABF, EBG, ABG, case, esac, mod, next, init, union, in, xor, xnor, self, TRUE, FALSE, count, abs, max, min

הגדרת משתני הקלט

כל סיגנל בתוכנית המקור ב- Esterel יקבל את הקידומת input בתוצר ה- NuSMV. זאת משתי סיבות:

- בשפת NuSMV ישנן מילים שמורות כגון A,S,X,Y,Z,O ועוד אותיות הנוטות לייצג סיגנלים בתוכניות Esterel. שימוש ב- prefix זה ימנע שגיאות קומפילציה בשפת היעד
- הוספת הקידומת input מוסיפה לקריאות התוצר ב- NuSMV

כל סיגנל יומר למשתנה IVAR בוליאני ב- NuSMV



הגדרת משתנה העזר AWAIT

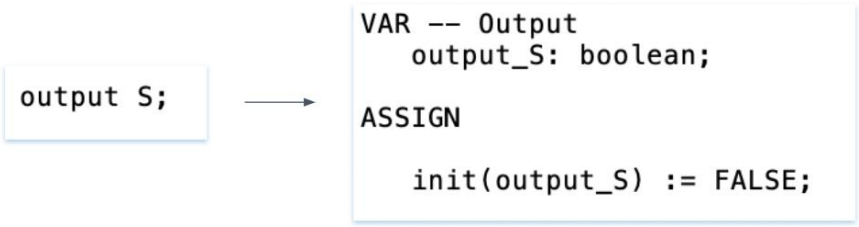
על מנת שנוכל להשתמש ביכולות של await נגדיר לכל סיגנל קלט S נגדיר פרמטר עזר הנקרא await_S והוא מיוצג באופן הבא:

```
MODULE main

IVAR -- Input
    input_S: boolean;

VAR -- await representation
    await_S: boolean;

ASSIGN

   init(await_S) := FALSE;

   next(await_S) := case
                    await_S : TRUE;
                    input_S : TRUE;
                    TRUE: FALSE;
                esac;
```

כלומר, ערכו של הפרמטר await הופך להיות TRUE ברגע שהסיגנל מתקבל בפעם הראשונה.

הגדרת משתני הפלט

בדומה להתמודדות עם סיגנלי הקלט, נוסיף את קידומת output לכל סיגנל פלט. כל סיגנל יומר למשתנה VAR בוליאני ב- NuSMV. ההנחה היא כי כל סיגנל קלט מאותחל לערך False.



הגדרת השעון הגלובלי ב- NUSMV

ייצוג שעון הגלובלי ב- NuSMV יראה כך:

```
VAR -- global clock representation
    global_clock : 0 .. 100;

ASSIGN

   init(global_clock) := 0;

   next(global_clock) := case
                  global_clock < 100: global_clock+1;
                  TRUE: global_clock;
              esac;
```

מפאת אילוצי ביצועים נאלצתי להגביל את השעון למספר סופי של צעדים (אחרת תוכנית ה- NuSMV נתקלת בקשיים)

התמודדות עם לולאות

ראשית, יש לזהות האם יש לולאות בתוכנית. ישנם שני סוגים:

1. לולאה אינסופית:

loop
  <statement>
end loop

2. לולאה אינסופית עם תנאי חזרה לתחילת הלולאה:

loop		**loop abort**
<statement>	≡	*<statement>* **; halt**
each *<delay>*		**when** *<delay>* **end loop**

האלגוריתם להתמודדות עם לולאות כולל את השלבים הבאים:

- זיהוי לולאות בתוכנית ה- Esterel וסוגם
- הגדרה ואתחול עבור כל לולאה:

```
VAR -- loops representation
    loop_state : {start, middle, end};
    loop_step : 0 .. 2;

ASSIGN

   init(loop_state) := start;
   init(loop_step) := 0;
```

- הגדרת מבנה ה- next עבור כל לולאה
- עבור כל סיגנל פלט המושפע מלולאה מסוימת יש לוודא כי הוא מתאפס ברגע שלולאה מסתיימת

נביט בדוגמא עבור לולאות:

```
module Foo:
input A,B;
output X;

loop
        emit X;
        await A;
        await B;
end loop
```

```
next(loop_state) := case
                      (await_B | input_B) & loop_step = 1 : end;
                      loop_state = start : middle;
                      loop_state = end : start;
                      TRUE: middle;
                    esac;

next(loop_step) := case
                      (await_A | input_A) & loop_step = 0 : loop_step+1;
                      (await_B | input_B) & loop_step = 1 : loop_step+1;
                      loop_step = 2 : 0;
                      TRUE: loop_step;
                    esac;


next(output_X) := case
                      loop_state = end : FALSE;
                      loop_state = start : TRUE;
                      TRUE: FALSE;
                   esac;
```

הגדרת NEXT עבור כל סיגנל פלט

עבור כל סיגנל פלט יש להגדיר את ערכו בכל אחד ממופעי השעון. המקבילה לכך ב-NuSMV הוא הייצוג ע"י next. לכן האלגוריתם להרכבת הייצוג של סיגנל הפלט הוא:

```
next_statement = " next(" + new_var_name + ") := case\n"
for input_line in lines:
        next_statement += match_line(input_line)
next_statement += "     TRUE: FALSE;\n"
next_statement += " esac;\n"
```

כאשר המעטפת זהה לכל הסיגנלים.

match_line היא הפונקציה אשר יודעת להמיר כל פקודה ב- Esterel לייצוג המתאים ב-NuSMV

התמודדות עם EMIT

ההמרה הפשוטה של הפקודה emit S מתרגמת לכך שערכו של next(s) יהיה TRUE.

עם זאת, יש לקחת בחשבון שרשור של אילוצים הנוגעים ללולאות, await, pause אשר צריכים להתקיים לפני פעולת ה- emit.

דוגמא עבור emit:

```
module Foo:
output X;

emit X;
```

→

```
MODULE main

VAR -- Output
   output_X: boolean;

ASSIGN

   init(output_X) := FALSE;

   next(output_X) := case
                        TRUE: TRUE;
                        TRUE: FALSE;
                     esac;
```

התמודדת עם PRESENT

המבנה של הפקודה present נראה כך:

present <*signal-expression*>
[**then** <*statement*> *]*
[**else** <*statement*> *]*
end present

כאשר then, else הם אופציונליים.

אלגוריתם ההמרה שולח את הצהרות ה- then ו- else לתרגום עצמאי (ברקורסיה), ומשרשר אותם להצהרת ה- next של הפלט הרלוונטי:

```
if (present_statement):
        then_sub_statement = match_line(then_statement)
        else_sub_statement = match_line(else_statement)

        next_statement += condition & then_sub_statement
        next_statement += ! condition & else_sub_statement
```

דוגמא עבור present:

```
module Foo:
input S;
output X,Y;


present S
        then emit X
        else emit Y
end present;
```

→

```
next(output_X) := case
                     input_S : TRUE;
                     TRUE: FALSE;
            esac;

next(output_Y) := case
                     ! input_S : TRUE;
                     TRUE: FALSE;
            esac;
```

התמודדת עם SEQUENCE

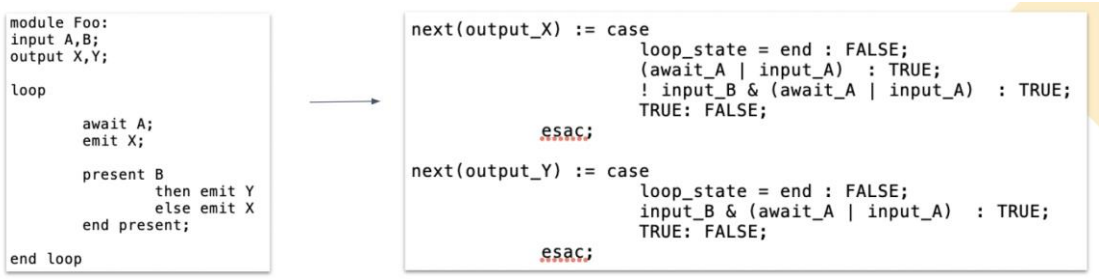
sequence הוא רצף של פקודות אחת אחרי השניה והוא נראה כך:
<statement_a> ; <statement_b>

כאשר תוכנית ההמרה נתקלת בפקודות מסוג זה היא שולחת כל אחת מהן להמרה נפרדת אך דואגת לשמר את התלויות ביניהן (כגון await).

לבסוף משרשרים הכל יחד:

```
if (sequence_statement):
        sub_statement_a = match_line(statement_a, dependencies)
        sub_statement_b = match_line(statement_b, dependencies)

        next_statement += sub_statement_a
        next_statement += sub_statement_b
```

דוגמא להמרה של sequence:



```
module Foo:
input A,B;
output X,Y;

loop

        await A;
        emit X;

        present B
                then emit Y
                else emit X
        end present;

end loop
```

```
next(output_X) := case
                        loop_state = end : FALSE;
                        (await_A | input_A)  : TRUE;
                        ! input_B & (await_A | input_A)  : TRUE;
                        TRUE: FALSE;
                esac;

next(output_Y) := case
                        loop_state = end : FALSE;
                        input_B & (await_A | input_A)  : TRUE;
                        TRUE: FALSE;
                esac;
```

התמודדות עם מקביליות

המבנה של פקודה מקבילית הוא: <statement_b> || <statement_a>

כאשר תוכנית ההמרה נתקלת בפקודות מקביליות היא שולחת את כל אחת מהן להמרה נפרדת ובלתי תלויה.

לבסוף משרשרים הכל יחד:

```
if (parallel_statement):
        sub_statement_a = match_line(statement_a)
        sub_statement_b = match_line(statement_b)

        next_statement += sub_statement_a
        next_statement += sub_statement_b
```

דוגמא להמרה עם מקביליות:

```
loop
        [
                await A;
                emit X;
        ||
                present B
                        then emit Y
                        else emit X
                end present;
        ]
end loop
```

```
next(output_X) := case
                        loop_state = end : FALSE;
                        (await_A | input_A)  : TRUE;
                        ! input_B & loop_state = start : TRUE;
                        TRUE: FALSE;
                esac;

next(output_Y) := case
                        loop_state = end : FALSE;
                        input_B & loop_state = start : TRUE;
                        TRUE: FALSE;
                esac;
```

התמודדות עם PAUSE

כאשר בתוכנית Esterel מופיעה הפקודה pause המשמעות היא שיש להמתין לסיבוב השעון הבא. כלומר, בסיבוב השעון הנוכחי לא תתבצע שום פעולה

הייצוג של pause ב- NuSMV נבנה בדרך הבאה:

- ראשית, מתבצעת סריקה של התוכנית כדי לזהות את המקומות בהם מופיעה הפקודה בכל תת תוכנית/ לולאה
- עבור כל תת תוכנית נוצרים שני משתנים ב - NuSMV:
  - pause_i : משתנה זה יהיה TRUE כאשר התוכנית צריך להיכנס להפסקה ו-FALSE אחרת.
  - pause_i_counter: בכל פעם שמתבצעת עצירה (pause=TRUE) המונה יתקדם ב-1. זאת על מנת שנוכל לסנכרן את שאר המשתנים לעצירה הנכונה.

דוגמא עם pause:

```
module Foo:
input A;
output X;


await A;
pause;
pause;
pause;
emit X;
```

```
next(pause) := case
                        (await_A | input_A) & pause_counter = 0 : TRUE;
                        pause_counter = 1 : TRUE;
                        pause_counter = 2 : TRUE;
                        TRUE: FALSE;
                esac;

next(pause_counter) := case
                        pause & pause_counter < 3 : pause_counter+1;
                        TRUE: pause_counter;
                esac;

next(output_X) := case
                        (await_A | input_A) & (pause_counter = 2)  : TRUE;
                        TRUE: FALSE;
                esac;
```

דוגמת המרה – ABRO

ABRO היא דוגמא מפורסמת בעולם השפות הסינכרוניות, נהוג לכנות אותה כ- Hello World של התכנות הסינכרוני. היא משמשת כדוגמא פשוטה להסביר את העקרונות של השפות השונות.

הגדרת מערכת ABRO:

- המערכת צריכה לפלוט את O כאשר היא מקבלת את הקלט A ו-B
- הקלט R צריך לגרום לאתחול המערכת

המימוש של ABRO בשפת Esterel נראה כך:

```
module ABRO:            // תכנית מורכבת ממודולים
input A,B,R;            //סיגנלי הקלט
output O;           //סיגנל הפלט
loop                // ללואה
    [await A || await B];     // עד אשר שני הסיגנלים יגיעו (המתנה במקביל)
    emit O                    // המערכת תפלוט את הסיגנל הפלט
when R                        // המערכת תפסיק את פעולתה בקבלת הסיגנל R
end module              // סוף התכנית
```

התנהגות המערכת מוצגת בטבלה הבאה:

Instant	Inputs	Outputs	Comment
0			
1	A,B	O	
2	A		Signal A ignored
3	R		Reset
4	A		Still waiting for B
5	B	O	
6	A,B,R		Reset, A and B ignored

כאשר נריץ את האלגוריתם שלנו על תוכנית ABRO נקבל את הפלט הבא:

```
MODULE main

IVAR -- Input
    input_A: boolean;
    input_B: boolean;
    input_R: boolean;

VAR -- await representation
    await_A: boolean;
    await_B: boolean;
    await_R: boolean;

VAR -- Output
   output_O: boolean;

VAR -- global clock representation
    global_clock : 0 .. 100;

VAR -- loops representation
    loop_state : {start, middle, end};
```

```
ASSIGN

   init(await_A) := FALSE;
   init(await_B) := FALSE;
   init(await_R) := FALSE;
   init(output_O) := FALSE;

   init(global_clock) := 0;

   init(loop_state) := start;


next(await_A) := case
                 loop_state = end : FALSE;
                 await_A : TRUE;
                 input_A : TRUE;
                 TRUE: FALSE;
             esac;
next(await_B) := case
                 loop_state = end : FALSE;
                 await_B : TRUE;
                 input_B : TRUE;
                 TRUE: FALSE;
             esac;
next(await_R) := case
                 loop_state = end : FALSE;
                 await_R : TRUE;
                 input_R : TRUE;
                 TRUE: FALSE;
             esac;

 next(global_clock) := case
                  global_clock < 100: global_clock+1;
                  TRUE: global_clock;
              esac;

 next(loop_state) := case
                  input_R: end;
                  loop_state = start : middle;
                  loop_state = end : start;
                  TRUE: middle;
              esac;

next(output_O) := case
                     loop_state = end : FALSE;
                     (await_A | input_A) & (await_B | input_B)  : TRUE;
                     TRUE: FALSE;
                  esac;
```

אימות באמצעות NUSMV

לאחר המרה של תוכנית Esterel ל- NuSMV אנו יכולים לבדוק האם תכונה (property) מסוימת מתקיימת על תוכנית הפלט.

הבטחת FAIRNESS

על מנת לבצע אימות עלינו לוודא כי כל סינגלי הקלט מתקבלים בצורה "הוגנת". עקרון ה-FAIRNESS מאפשר זאת.

לכן, עבור כל סיגנל נוסיף את ההצהרה FAIRNESS input_Signal.

המרת תכונות לבדיקה

תוכנית ה- NuSMV שמתקבלת שומרת על המבנה הלוגי של תוכנית Esterel המקורית. בהתאם למבנה ה- next של NuSMV, התגובתיות של התוכנית אינה מיידית אלא מתבצעת רק בצעד הבא. לכן, כאשר ברצוננו לאמת תכונות עלינו לוודא כי הם מתייחסים לעובדה זו.

למשל ב-ABRO נרצה לוודא כי: כאשר A,B מתקבלים ו- R לא מתקיים אז O יהפוך לאמת. נבטא זאת כך:

```
check_ltlspec -p "G (input_A & input_B ->  X output_O)"
```

דוגמא א' – ABRO

נאמת כי כאשר A,B מתקבלים ו- R לא מתקיים אז O יהפוך לאמת

```
NuSMV > read_model -i /Users/tehila/Desktop/project/ABRO_Esterel.strl20061_NuSMV.smv
NuSMV > flatten_hierarchy
NuSMV > encode_variables
NuSMV > build_model
NuSMV > check_ltlspec -p "G (input_A & input_B & !input_R ->  X output_O)"
-- specification  G (((input_A & input_B) & !input_R) ->  X output_O)  is true
NuSMV > check_ltlspec -p "G ((input_A & (!input_R U input_B)) -> (!input_R U ( X output_O)))"
-- specification  G ((input_A & (!input_R U input_B)) -> (!input_R U ( X output_O)))  is true
NuSMV > check_ltlspec -p "G ((input_B & (!input_R U input_A)) -> (!input_R U ( X output_O)))"
-- specification  G ((input_B & (!input_R U input_A)) -> (!input_R U ( X output_O)))  is true
```

דוגמא ב'

נביט בדוגמא נוספת עבור תוכנית ה- Esterel הבאה:

```
module Foo:
input A,R;
output X,Y,Z;
loop
        [
                await A;
                emit X;
        ||
                present X
                        then emit Y
                        else emit Z
                end present;
        ]
each R
end module
```

הפלט של האלגוריתם שלנו יניב את קוד ה- NuSMV הבא:

```
next(output_X) := case
                    (await_A | input_A)  : TRUE;
                    loop_state = end : FALSE;
                    TRUE: FALSE;
            esac;

next(output_Y) := case
                    output_X : TRUE;
                    loop_state = end : FALSE;
                    TRUE: FALSE;
            esac;

next(output_Z) := case
                    ! output_X : TRUE;
                    loop_state = end : FALSE;
                    TRUE: FALSE;
            esac;
```

כעת ניתן לאמת כי המצאות הסיגנל A גוררת את הימצאותו של Y

- (בהינתן A => סיגנל הפלט X יהפוך לאמת בצעד הבא
- בהינתן X => סיגנל הפלט Y יהפוך לאמת בצעד הבא)

```
NuSMV > read_model -i /Users/tehila/Desktop/project/Example22_Esterel.strl38302_NuSMV.smv
NuSMV > flatten_hierarchy
NuSMV > encode_variables
NuSMV > build_model
NuSMV > check_ltlspec -p "F(output_Y)"
-- specification  F output_Y  is true
NuSMV > check_ltlspec -p "F(output_Z)"
-- specification  F output_Z  is true
NuSMV > check_ltlspec -p "G(output_X -> X output_Y)"
-- specification  G (output_X ->  X output_Y)  is true
NuSMV > check_ltlspec -p "G(input_A -> X X output_Y)"
-- specification  G (input_A ->  X ( X output_Y))  is true
```

קוד הפרויקט

```
#!/usr/bin/env python3

import re
import sys
import random
from datetime import datetime



## This project converts Esterel source code to NuSMV source code
## Author: Tehila Shneider

def writeline(string, file):
    file.write(string + "\n")

def handle_module_defintion(string):
    #line = re.sub("module","MODULE", string)
    #line = line.replace(':', '')

    line = "MODULE main\n"
    return line

#spliting the Esterel vars definition to a list of vars
def split_to_vars(string):

        if string.startswith("input"):
            string = string[len("input"):]

        if string.startswith("output"):
            string = string[len("output"):]

        string = string[:-len("\n")]
        string = string.replace(' ', '')
        string = string.replace(';', '')

        vars = string.split(',')

        return vars


#handling the input string on the Esterel program
def handle_input_definition(string):
    #split into vars
    vars = split_to_vars(string)

    #init input block with IVAR
    input_block = "IVAR -- Input \n"
```

```
    # all vars in the input definition are signals translated to boolean input
variables in NuSMV
    for var in vars:
        input_block += "    input_" + var + ": boolean;\n"
        input_var_list.append("input_" + var)
        original_input_var_list.append(var)

    return input_block


#creating await vars for the input string on the Esterel program
def create_await_definition(string):
    #split into vars
    vars = split_to_vars(string)

    await_block = "VAR -- await representation \n"

    #In order to support the await fundtion each input var will have a
corresponding await var
    for var in vars:
        await_block += "    await_" + var + ": boolean;\n"
        await_var_list.append("await_" + var)

    return await_block


#handling the output string on the Esterel program
def handle_output_definition(string):
    #split into vars
    vars = split_to_vars(string)

    #init output block with VAR
    output_block = "VAR -- Output \n"

    # all vars in the output definition are translated to boolean variables in
NuSMV
    for var in vars:
            output_block += "   output_" + var + ": boolean;\n"
            output_var_list.append("output_" + var)
            original_output_var_list.append(var)

    return output_block


#creating the init statement for all await and output variables
def init_all_vars():
    init_string = ""

    #all await variables should be initialized to false
```

```
    for await_var in await_var_list:
        init_string += "    init(" + await_var + ") := FALSE;\n"

    #all await variables should be initialized to false
    for output_var in output_var_list:
        init_string += "    init(" + output_var + ") := FALSE;\n"

    return init_string


# creaing a next block to represent each await variable
def create_next_await_block(is_loop, loop_name):
        next_await_block = ""
        index = 0;

        for await_var in await_var_list:
            input_var = input_var_list[index]
            next_await_block += "    next(" + await_var + ") := case\n"
            next_await_block += "                " + input_var + " : TRUE;\n"

            #end of loop should reset await to FALSE
            if (is_loop):
                next_await_block += "                " + loop_name + " = end :
FALSE;\n"

            next_await_block += "                " + await_var + " : TRUE;\n"
            next_await_block += "                " + "TRUE: FALSE;\n"
            next_await_block += "            esac;\n"
            index += 1

        return next_await_block


# creaing a next block for each output variable
def create_next_statement_for_output_var(file_in_one_string,
original_var_name, new_var_name, is_loop, loop_name):
    var_next_statement = "    next(" + new_var_name + ") := case\n"


    in_file = open(sys.argv[1], 'r')
    lines = file_in_one_string.split(';')

    inner_pause_counter = -1
    await_detected_list = list()

    def match_line(sub_line, sub_await_list):
        full_await_string = ""
        full_await_string = concatenate_await_statement(sub_await_list)
        nonlocal inner_pause_counter
        partial_next_statement = ''
```

```
        #detect and handle parallel statement
        if
(re.fullmatch("(?P<process_a>[\w,\W]*\S+[\w,\W]*)\|\|(?P<process_b>[\w,\W]*\
S+[\w,\W]*)", sub_line)):
            parallel_match =
re.fullmatch("(?P<process_a>[\w,\W]*\S+[\w,\W]*)\|\|(?P<process_b>[\w,\W]*\
S+[\w,\W]*)", sub_line)
            process_a = parallel_match.group('process_a')
            process_b = parallel_match.group('process_b')

            sub_await_list_a = sub_await_list.copy()
            sub_await_list_b = sub_await_list.copy()

            sub_statement_a = match_line(process_a, sub_await_list_a)
            sub_statement_b = match_line(process_b, sub_await_list_b)

            if (sub_statement_a):
                partial_next_statement += sub_statement_a
            if (sub_statement_b):
                partial_next_statement += sub_statement_b

        #detect and handle sequence statements
        elif
(re.fullmatch("(?P<statement_a>[\w,\W]*\S+[\w,\W]*)\;(?P<statement_b>[\w,\
W]*\S+[\w,\W]*)", sub_line)):
            parallel_match =
re.fullmatch("(?P<statement_a>[\w,\W]*\S+[\w,\W]*)\;(?P<statement_b>[\w,\W
]*\S+[\w,\W]*)", sub_line)

            statement_a = parallel_match.group('statement_a')
            statement_b = parallel_match.group('statement_b')

            sub_statement_a = match_line(statement_a, sub_await_list)
            sub_statement_b = match_line(statement_b, sub_await_list)

            if (sub_statement_a):
                partial_next_statement += sub_statement_a
            if (sub_statement_b):
                partial_next_statement += sub_statement_b

        #detect and handle await statement
        elif (re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*",
sub_line)):
            await_match =
re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*", sub_line)
            new_await_var = await_match.group('new_await_var')
            if is_new_var_detected(await_detected_list,new_await_var):
                await_detected_list.append(new_await_var)
            if is_new_var_detected(sub_await_list, new_await_var):
```

```
            sub_await_list.append(new_await_var)

        #detect and handle emit statement
        elif (re.match("\s*emit\s+"+original_var_name+"\s*", sub_line)):

            if (len(full_await_string)):
                partial_next_statement += full_await_string

            if (inner_pause_counter>-1):
                partial_next_statement += "& (pause_counter = " +
str(inner_pause_counter) + ") "
            if (partial_next_statement):
                partial_next_statement += " : TRUE;\n"
            else:
                partial_next_statement += "TRUE: TRUE;\n"

        #detect and handle sustain statement
        elif (re.match("\s*sustain\s+"+original_var_name+"\s*", sub_line)):
            partial_next_statement += "    "
            if (len(full_await_string)):
                partial_next_statement += full_await_string
            if (inner_pause_counter>-1):
                partial_next_statement += "& (pause_counter = " +
str(inner_pause_counter) + ") "
            if (partial_next_statement):
                partial_next_statement += " : TRUE;\n"
            else:
                partial_next_statement += "TRUE: TRUE;\n"

        #detect and handle "present <signal-expression> [ then <statement> ] [
else <statement> ] end present" statement
        elif
(re.fullmatch("\s*present\s+(?P<present_signal>\w+)\s+then\s+(?P<then_stat
ement>\w+[\w, ]*)\s+else\s+(?P<else_statement>\w+[\w,
]*)\s+end\s+present\s*\;*\s*", sub_line)):
            present_then_match
=re.fullmatch("\s*present\s+(?P<present_signal>\w+)\s+then\s+(?P<then_stat
ement>\w+[\w, ]*)\s+else\s+(?P<else_statement>\w+[\w,
]*)\s+end\s+present\s*\;*\s*", sub_line)
            present_signal = present_then_match.group('present_signal')
            then_statement = present_then_match.group('then_statement')
            else_statement = present_then_match.group('else_statement')

            then_sub_statement = match_line(then_statement,
sub_await_list.copy())

            signal_name = convert_signal_name(present_signal)
            if (then_sub_statement == "TRUE: TRUE;\n"):
                partial_next_statement += signal_name + " : TRUE;\n"
            elif then_sub_statement:
```

```
            partial_next_statement += signal_name + " &" +
then_sub_statement

        else_sub_statement = match_line(else_statement,
sub_await_list.copy())
        if (else_sub_statement == "TRUE: TRUE;\n"):
            partial_next_statement += "    ! " + signal_name + " : TRUE;\n"
        elif else_sub_statement:
            partial_next_statement += "    ! " + signal_name + " &" +
else_sub_statement


    #detect and handle "present <signal-expression> [ then <statement> ]
end present" statement
    elif
(re.fullmatch("\s*present\s+(?P<present_signal>\w+)\s+then\s+\w+[\w,
]*\s+end\s+present\s*\;*\s*", sub_line)):
        present_then_match
=re.fullmatch("\s*present\s+(?P<present_signal>\w+)\s+then\s+(?P<then_stat
ement>\w+[\w, ]*)\s+end\s+present\s*\;*\s*", sub_line)
        present_signal = present_then_match.group('present_signal')
        then_statement = present_then_match.group('then_statement')

        signal_name = convert_signal_name(present_signal)
        sub_statement = match_line(then_statement, sub_await_list.copy())
        if (sub_statement    == "TRUE: TRUE;\n"):
            partial_next_statement += signal_name + " : TRUE;\n"
        elif sub_statement:
            partial_next_statement += signal_name + " &" + sub_statement

    #detect and handle pause statement
    elif (re.match(r"\s*pause\s*\;*", sub_line)):
        inner_pause_counter +=1

    #detect and handle " if <Boolean-expression> then <statement> [ else
<statement> ] end if"

elif(re.fullmatch("\s*if\s+(?P<if_statement>\w+)\s+then\s+(?P<then_statement
>\w+[\w, ]*)\s+else\s+(?P<else_statement>\w+[\w,
]*)\s+end\s+present\s*\;*\s*", sub_line)):
        if_then_match =
re.fullmatch("\s*if\s+(?P<if_statement>\w+)\s+then\s+(?P<then_statement>\w
+[\w, ]*)\s+else\s+(?P<else_statement>\w+[\w, ]*)\s+end\s+present\s*\;*\s*",
sub_line)
        if_statement = if_then_match.group('if_statement')
        then_statement = if_then_match.group('then_statement')
        else_statement = if_then_match.group('else_statement')

        then_sub_statement = match_line(then_statement,
sub_await_list.copy())
```

```
            if then_sub_statement:
                partial_next_statement += "    " + if_statement + " &" +
then_sub_statement

            else_sub_statement = match_line(else_statement,
sub_await_list.copy())
            if else_sub_statement:
                partial_next_statement += "    ! " + if_statement + " &" +
else_sub_statement

        #detect and handle " if <Boolean-expression> then <statement> end if"

elif(re.fullmatch("\s*if\s+(?P<if_statement>\w+)\s+then\s+(?P<then_statement
>\w+[\w, ]*)\s+end\s+present\s*\;*\s*", sub_line)):
            if_then_match =
re.fullmatch("\s*if\s+(?P<if_statement>\w+)\s+then\s+(?P<then_statement>\w
+[\w, ]*)\s+end\s+present\s*\;*\s*", sub_line)
            if_statement = if_then_match.group('if_statement')
            then_statement = if_then_match.group('then_statement')

            then_sub_statement = match_line(then_statement,
sub_await_list.copy())
            if then_sub_statement:
                partial_next_statement += "    " + if_statement + " &" +
then_sub_statement

        return partial_next_statement


    #for input_line in lines:
    for input_line in in_file.readlines():

        match_line_statement = match_line(input_line, await_detected_list)
        var_next_statement += match_line_statement

    if (is_loop):
        var_next_statement += "    " + loop_name + " = end : FALSE;\n"
    var_next_statement += "    TRUE: FALSE;\n"
    var_next_statement += "            esac;\n"
    return var_next_statement


#checking if variable is already in list
def is_new_var_detected(list, var):
    for element in list:
        if(var == element) :
            return False
    return True
```

```
# converting a variable in the Esterel program to the corresponding varibale in
the NuSMV program
def convert_signal_name(signal):

    for element in original_input_var_list:
        if(signal == element) :
            return "    input_" + signal

    else:
        return "    output_" + signal


#creating the await staement for the detected list
def concatenate_await_statement(await_detected_list):
    statement = ""
    i = 0
    for await_var in await_detected_list:
        await_sub_string = "(await_" + await_var + " | " + "input_" + await_var +
") "
        if (i == 0):
                statement += await_sub_string
        else:
            statement += "& " + await_sub_string
        i +=1
    return statement


#creating the global clock definition
def create_global_clock():
    clock_max = "1"
    clock_list = list()

    clock_var = "VAR -- global clock representation \n"
    clock_var += "    global_clock : 0 .. " + clock_max + ";\n"

    clock_init = "   init(global_clock) := 0;\n"

    clock_next = "   next(global_clock) := case\n"
    clock_next += "                global_clock < "+ clock_max +":
global_clock+1;\n"
    clock_next += "                TRUE: global_clock;\n"
    clock_next += "            esac;\n"

    clock_list.append(clock_var)
    clock_list.append(clock_init)
    clock_list.append(clock_next)

    return clock_list
```

```
# creating the statement defining the end of a loop
def create_end_loop_statement(file_string, loop_step):
    lines = file_string.split('\n')
    loop_step_statement = ""
    end_loop_statement = ""
    counter =0
    await_detected_list = list()

    for line in lines:
        if (re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*",line)):
            await_match =
re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*", line)
            new_await_var = await_match.group('new_await_var')
            signal_to_wait_for = concatenate_await_statement([new_await_var])
            loop_step_statement += "                    " + signal_to_wait_for + "&
loop_step = " + str(counter)+ " : loop_step+1;\n"

            #detecting last step to end the loop
            if (counter == loop_step-1):
                end_loop_statement += "                    " + signal_to_wait_for + "&
loop_step = " + str(counter)+ " : end;\n"
            counter +=1

    return end_loop_statement, loop_step_statement

#creating the representation of a loop (var, init, next)
def handle_end_loop(file_string, loop_condition):
    #list representing the end of a loop details
    end_loop_list = list()
    loop_step = len(re.findall(r"await", file_string))

    loop_condition = loop_condition.split('\n')
    condition = loop_condition[0].replace(' ', '')

    #VAR block for the end loop statement
    end_loop_var = "VAR -- loops representation \n"
    end_loop_var += "    loop_state : {start, middle, end};\n"
    end_loop_var += "    loop_step : 0 .. " + str(loop_step) + ";\n"


    end_loop_init = "   init(loop_state) := start;\n"
    end_loop_init += "  init(loop_step) := 0;\n"

    end_loop_statement, loop_step_statement =
create_end_loop_statement(file_string, loop_step)

    end_loop_next = "   next(loop_state) := case\n"

    if condition:
        end_loop_next += "                    input_" + condition + ": end;\n"
```

```
    else:
        end_loop_next += end_loop_statement

    end_loop_next += "                loop_state = start : middle;\n"
    end_loop_next += "                loop_state = end : start;\n"
    end_loop_next += "                " + "TRUE: middle;\n"
    end_loop_next += "            esac;\n"

    end_loop_next += "    next(loop_step) := case\n"
    end_loop_next += loop_step_statement
    end_loop_next += "                loop_step = " + str(loop_step) + " : 0;\n"
    end_loop_next += "                "+ "TRUE: loop_step;\n"
    end_loop_next += "            esac;\n"

    end_loop_list.append(end_loop_var)
    end_loop_list.append(end_loop_init)
    end_loop_list.append(end_loop_next)

    return end_loop_list


#detecting if the Esterel program contains loop and handling them
def detect_loops(file_string):
    #every element in the list should be (loop_var, init_loop,
next_statement_loop)
    if
(re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\W]*)each(?P
<loop_condition>[\w,\W]*\S+[\w,\W]*)", file_string)):

loop_match=re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\
W]*)each(?P<loop_condition>[\w,\W]*\S+[\w,\W]*)", file_string)
        loop_body = loop_match.group('loop_body')
        loop_condition = loop_match.group('loop_condition')
        end_loop_list = handle_end_loop(file_string, loop_condition)
        is_loop = True
        end_loop_var = end_loop_list[0]
        end_loop_init = end_loop_list[1]
        end_loop_next = end_loop_list[2]

    elif
(re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\W]*)when(?P
<loop_condition>[\w,\W]*\S+[\w,\W]*end\s+loop[\w,\W]*\S+[\w,\W]*)",
file_string)):
        loop_match =
re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\W]*)when(?P
<loop_condition>[\w,\W]*\S+[\w,\W]*end\s+loop[\w,\W]*\S+[\w,\W]*)",
file_string)
        loop_condition = loop_match.group('loop_condition')
        end_loop_list = handle_end_loop(file_string, loop_condition)
```

```
    elif
(re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\W]*)end\s+lo
op[\w,\W]*", file_string)):
        loop_match =
re.match("[\s,\[,\],\w,\W]*loop\s+(?P<loop_body>[\w,\W]*\S+[\w,\W]*)end\s+loo
p[\w,\W]*", file_string)
        end_loop_list = handle_end_loop(file_string, "")

    else:
        end_loop_list = list()
    return end_loop_list


#creating the next definition for pause
def create_pause_next(file_string, pause_counter):
    lines = file_string.split('\n')
    pause_next = ""
    counter =0
    await_detected_list = list()

    for line in lines:
        if (re.fullmatch(r"\s*pause\s*\;*", line)):
            signals_before_pause =
concatenate_await_statement(await_detected_list)
            if signals_before_pause:
                pause_next += "                  " + signals_before_pause + "& "
            pause_next += "pause_counter = " + str(counter)+ " : TRUE;\n"
            await_detected_list = list()
            counter +=1
        elif (re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*",line)):
            await_match =
re.match("[\s,\[,\]]*await\s+(?P<new_await_var>\w+)[\s,\[,\]]*", line)
            new_await_var = await_match.group('new_await_var')
            if is_new_var_detected(await_detected_list,new_await_var):
                await_detected_list.append(new_await_var)

    return pause_next


#detecting pauses and handling them
def handle_pause_statements(file_string):
    pause_list = list()
    pause_counter = len(re.findall(r"pause", file_string))

    #there are pause statements in the programs and we should handle them
    if (pause_counter > 0):
        pause_vars = "VAR -- pause representation \n"
        pause_vars += "    pause : boolean;\n"
        pause_vars += "    pause_counter : 0 .. " + str(pause_counter) + ";\n"
```

```
        pause_init = "   init(pause) := FALSE;\n"
        pause_init += "   init(pause_counter) := 0;\n"

        pause_next = "   next(pause) := case\n"
        pause_next += create_pause_next(file_string, pause_counter)
        pause_next += "                  " + "TRUE: FALSE;\n"
        pause_next += "              esac;\n"

        pause_next += "   next(pause_counter) := case\n"
        pause_next += "                  " + "pause & pause_counter < " +
str(pause_counter) + " : pause_counter+1;\n"
        pause_next += "                  "+ "TRUE: pause_counter;\n"
        pause_next += "              esac;\n"

        pause_list.append(pause_vars)
        pause_list.append(pause_init)
        pause_list.append(pause_next)


    return pause_list


#creating the fairness declaration for each input variable
def create_fairness_block():
    fairness_block = ""

    for var in input_var_list:
        fairness_block += "FAIRNESS " + var + ";\n"

    return fairness_block


#main logic to convert the Esterel program to NuSMV
def convert():
    # set input and output files
    random_int = str(random.randrange(100000))
    out_name = sys.argv[1] + random_int + "_NuSMV.smv"
    out_file = open(out_name, 'w')

    in_file = open(sys.argv[1], 'r')

    module_def = ""
    VAR_block = ""
    init_block = ""
    next_block = ""
    next_await_block = ""

    end_loop_var = ""
    end_loop_init = ""
```

```
end_loop_next = ""
end_loop_list = list()
loop_name = "loop_state"


in_file_to_string = open(sys.argv[1], 'r')
file_in_one_string =""
for line in in_file_to_string.readlines():
    file_in_one_string += line
#print(file_in_one_string)

#create global clock representation
clock_list = create_global_clock()
clock_var, clock_init, clock_next = clock_list

#detect if program contains loops with conditions
end_loop_list = detect_loops(file_in_one_string)
if (len(end_loop_list)>0):
    is_loop = True
    end_loop_var, end_loop_init, end_loop_next = end_loop_list

else:
    is_loop = False

#detect and handle pause statements
pause_list = handle_pause_statements(file_in_one_string)
if (len(pause_list) == 3):
    pause_vars, pause_init, pause_next = pause_list
else:
    pause_vars = pause_init = pause_next = ""

prev_count = 0
for input_line in in_file.readlines():
    #detect blank lines
    test_line = input_line.strip()
    if len(test_line) == 0:
        pass

    else:
        if  (input_line.startswith("module")):
            module_def += handle_module_defintion(input_line)

        elif (input_line.startswith("input")):
            split_to_vars(input_line)
            VAR_block += handle_input_definition(input_line)
            VAR_block += "\n" + create_await_definition(input_line)

        elif (input_line.startswith("output")):
            VAR_block += "\n" + handle_output_definition(input_line)
        else:
```

```
                new_line = input_line

    output_var_index=0


    init_block = init_all_vars()
    next_await_block = create_next_await_block(is_loop, loop_name)

    #all input variables are considered to be fair so validation would be possible
    fairness_block = create_fairness_block()


    #build_file

    #VARS section
    writeline(module_def, out_file)
    writeline(VAR_block, out_file)
    writeline(clock_var, out_file)
    writeline(end_loop_var,out_file)
    writeline(pause_vars, out_file)
    writeline(fairness_block, out_file)

    writeline("\nASSIGN\n", out_file)

    #init section
    writeline(init_block, out_file)
    writeline(clock_init, out_file)
    writeline(end_loop_init, out_file)
    writeline(pause_init, out_file)

    #next section
    writeline(next_await_block, out_file)
    writeline(clock_next, out_file)
    writeline(end_loop_next, out_file)
    writeline(pause_next, out_file)

    for output_var_name in original_output_var_list:
        new_var_name = output_var_list[output_var_index]
        next_statement_for_i =
create_next_statement_for_output_var(file_in_one_string, output_var_name,
new_var_name, is_loop, loop_name)
        output_var_index +=1
        writeline(next_statement_for_i, out_file)

    # file is done, inform user
    print("Done converting Esterel program!")

def print_file(file):
    open(file, 'r')
    for line in file.readlines():
```

```
        print(line)

#define gloabl vraiables lists

#original input vars in the Esterel program
original_input_var_list = list()

#original output vars in the Esterel program
original_output_var_list = list ()


#the representation for the input vars in the NuSMV program
input_var_list = list()

#the representation for the await var for each input var
await_var_list = list()

#the representation for the output vars in the NuSMV program
output_var_list = list()


if __name__ == '__main__':
    if len(sys.argv) != 2:
        print("usage: python3 esterel2nusmv.py
ESTEREL_PROGRAM_TO_CONVERT.smv")
        exit

    convert()
```

רשימת מקורות

[1] Kenneth L. McMillan, Symbolic Model Checking, 1993

[2] Edmund M. Clarke, Allen E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic". Logic of Programs, Proceedings of Workshop, Lecture Notes in Computer Science, Vol. 131. Springer, Berlin: 52–71, 1981.

[3] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, Marco Roveri, "NuSMV: a new Symbolic Model Verifier", CAV '99 Proceedings of the 11th International Conference on Computer Aided Verification, Pages 495 - 499, 1999.

[4] Albert Benveniste; Gérard Berry, "The synchronous approach to reactive and real-time systems", Proc. IEEE, vol. 79, pp. 1270–1282, Sept. 1991.

[5] Amar Bouali, “Xeve: An Esterel verification environment,” in Proc 10th Int. Conf. Comput.-Aided Verification (CAV ’98), vol. 1427, LNCS, Vancouver, BC, 1998.

[6] Axel Poigné, Matthew Morley, Olivier Maffeïs, Leszek Holenderski, Reinhard Budde: "The Synchronous Approach to Designing Reactive Systems", Formal Methods in System Design 12(2): 163-187 (1998)

[7] Monika Müllerburg, Leszek Holenderski, Olivier Maffeïs, Agathe Merceron, Matthew Morley: Systematic testing and formal verification to validate reactive programs. Software Quality Journal 4(4): 287-307 (1995)

[8] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, Stefano Tonetta, "The nuXmv Symbolic Model Checker", International Conference on Computer Aided Verification, CAV 2014: Computer Aided Verification pp 334-342.

[9] Gérard Berry, The Constructive Semantics of Pure Esterel, 1999.

[10] Gordon D. Plotkin, “A structural approach to operational semantics,” Aarhus University, Åarhus, Denmark, DAIMI FN-19, 1981.

[11] Janusz A. Brzozowski, “Derivatives of regular expressions,” J. Assoc. Comput. Mach., vol. 11, pp. 481–494, Oct. 1964.

[12] Georges Gonthier, “Sémantiques et Modéles d’Exècution des Langages Rèactifs Synchrones: Application Esterel. [Semantics and Models of Execution of the Synchronous Reactive Languages: Application to Esterel],” These d’Informatique, Universite d’Orsay, Orsay, France, 1988.

[13] Sharad Malik, “Analysis of cyclic combinational circuits,” IEEE Trans. Computer-Aided Design, vol. 13, pp. 950–956, July 1994.

[14] Stephen A. Edwards, “An Esterel compiler for large control-dominated systems,” IEEE Trans. Computer-Aided Design, vol. 21, Feb. 2002.

[15] Daniel Weil, Valerie Berlin, Etienne Closse, Michel Poize, Patrick Venier, and Jacques Pulou, “Efficient compilation of Esterel for real-time embedded systems,” in Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Syst. (CASES), San Jose, CA, Nov. 2000, pp. 2–8.

[16] R. S. French, M. S. Lam, J. R. Levitt, and K. Olukotun. A general method for compiling event-driven simulations. Presented at 32nd Design Automation Conference. [Online]. Available: http://suif.stanford.edu/papers/rfrench95.ps