מכון ויצמן למדע
WEIZMANN INSTITUTE OF SCIENCE

The Rothschild-Weizmann Program for Excellence in Science Teaching
Funded by the Caesarea Edmond Benjamin de Rothschild Foundation

בס״ד

ל׳ תשרי התשע״ג 16/10/2012

פרויקט גמר

בנושא

Model checking

אימות נוסחאות LTL

ובניית מודל אוטומט LGBA

עפ״י האלגוריתם של

ורדי-פלד-גרט-וולפר

"Simple On-the-fly Automatic Verification of Linear Temporal Logic"

מגיש:

עודד פרץ

ת.ז.: 037690187

מנחה:

פרופ׳ מוטי בן-ארי

תוכן עניינים

מבוא.

תחום אימות התוכנה- Software Verification , עוסק בשיטות להוכחת נכונותן של תוכניות, מערכות אוטומטיות, מעגלים חשמליים וכדומה. מערכות אוטומטיות ניתן לסווג לשני סוגים עיקריים: הסוג הראשון הוא מערכת שכאשר היא מקבלת קלט מסוים היא מתחילה חישוב ואנו מצפים שהחישוב יגיע לידי מיצוי ויסתיים בסופו של תהליך. הסוג השני הוא מערכות אשר ברגע שהן מתחילות את עבודתן אנו מצפים שהן תמשכנה לעבוד ולרוץ כל הזמן, כדוגמת מערכות הפעלה ובקרה. לכל אחד מסוגי המערכות האלו יש שיטה לבדיקת תקינות המערכת. מערכות חישוביות בד"כ מאומתות ע"י לוגיקה של תחשיב הפסוקים, ועליהם לא ארחיב בעבודתי זו. סוג המערכות השני מאומתות בד"כ בשיטת ה- model checking , כלומר ע"י מודלים חישוביים וגרפים המשתמשים בלוגיקה טמפורלית (עיתית-תלוית זמן). מודלים אלו ידמו מערכות הפועלות ללא הפסק, אך בעלות **מספר סופי** של מצבים והפעלתם תאמת אם התכנית תקינה (מהי תקינות אגדיר בהמשך) או לא. כאשר יש בעיה במערכת המודל יציג אותה. השפה העיקרית בה משתמשים במודלים מעין אלו היא שפת הלוגיקה הטמפורלית LTL: Linear Temporal Logic.

בעבודתי זו אציג את הגדרת הלוגיקה הטמפורלית ואדגים מספר נוסחאות בשפה, כמו כן אציג את הגדרת האוטומט LGBA ואציג מספר דוגמאות של אוטומטים אלו. לאחר מכן אציג בקצרה של מאפייני ה-Model Checking ולאחר מכן אציג את האלגוריתם של ורדי-פלד-גרט-וולפר לאימות בזמן אמת של נוסחא ב LTL ומציאת אוטומט (LGBA) מתאים לנוסחא. לאחר הצגת האלגוריתם והסבר מפורט על דרך פעולתו אציג מספר דוגמאות לבדיקת נוסחאות ב LTL ויצירת אוטומט עבורן. לסיום אציג מספר שיפורים שהוצעו ע"י מפתחי האלגוריתם ואציע מספר שיפורים חדשים.

1. הלוגיקה הקלאסית ושפת ה- LTL.

1.1.הלוגיקה הקלאסית

הלוגיקה המתמטית הקלאסית פותחה בכדי לאפשר ייצוג בצורת נוסחאות של טענות לוגיות. שפה זו מכילה "אטומים" שהם פסוקים/טענות בסיסיות, המיוצגים ע"י אותיות לטיניות כגון p,q,a,b וכדומה, וכן קשרים לוגיים משני סוגים: קשרים בינאריים כגון: $(\wedge,\vee,\rightarrow,\leftrightarrow)$ המבטאים קשר בין שני אטומים או נוסחאות, וכן קשר אונארי : $\neg$ המבטא את שלילת האטום או הנוסחא, למשל $\neg p$ מציין כי p לא מתקיים. קיימים קשרים בינאריים נוספים, אך אלו שציינתי הם השימושיים ביותר. לוגיקה מתקדמת יותר מכילה בנוסף לתחשיב הפסוקים הנ"ל גם 2 כמתים: $\forall$ ו- $\exists$ שמשמעותם היא : "לכל" ו-"קיים" בהתאמה. כמתים אלו מוסיפים לתחשיב הפסוקים הבסיסי את היכולת לדבר על כמויות כגון "לכל a , a גורר את b" או "קיים a שעבורו a גורר את b". על בסיס לוגיקה זו פיתח אמיר פנואלי את לוגיקת הזמנים הליניארית, ה-LTL.

1.2.שפת ה- Linear Temporal Logic -LTL

כאמור על בסיסה של הלוגיקה הקלאסית פיתח פרופ׳ אמיר פנואלי בשנת 1977 את לוגיקת הזמנים הליניארית, ה-LTL, שפה שמאפשרת לנו לעסוק ולייצג גם את זמן התרחשות הפסוקים, זאת ע״י הוספת אופרטורים-קשרים לוגיים המתייחסים לזמן.

הגדרה פורמאלית של שפת ה-LTL :

הגדרת שפת ה-LTL נעשית באופן אינדוקטיבי על בסיס הלוגיקה הקלאסית:

- קיימים אטומים מתוך קבוצה סופית D .קבוצה D היא המקור עבור שפת ה-LTL.
- כל אטום הוא נוסחא.
- עבור נוסחאות φ,ψ מתקיים: $\Box\varphi$, $\varphi\bigcup\psi$, $\varphi\vee\psi$, $\varphi\wedge\psi$, $\neg\varphi$ הן נוסחאות, (הסמנטיקה והפירוש של האופרטורים $\Box$, $\bigcup$ תוצג בהמשך).

כמו בלוגיקה הקלאסית שבה מעבר לאופרטורים הבסיסים המספיקים (למשל קבוצת הקשרים המספיקה: $\{\neg,\wedge\}$) הוגדרו אופרטורים נוספים ע״מ להעשיר את השפה ולפשט את הצגת הנוסחאות בשפה, גם ה-LTL הוגדרו אופרטורים נוספים פרט ל $\Box$, $\bigcup$ שהוצגו בהגדרה , כגון V , $\Diamond$, $\Box$ וזאת על-מנת להעשיר ולפשט את כתיבת הנוסחאות בשפה.

הקשרים הטמפורליים מיוצגים הן ע״י סימן ו/או אות לטינית גדולה, והם נחלקים לקשרים בינאריים וקשרים אונאריים.

להלן טבלת הקשרים הטמפורליים השימושיים ביותר:

סוג הקשר	סימן	ייצוג טקסטואלי	משמעות האות	הסבר
בינארי	$\phi U\psi$	$\phi U\psi$	**U**ntil	ϕ יתקיים עד ש ψ יתקיים (ψ <u>חייב</u> להתקיים בסוף, ייתכן ש ψ יתקיים בהתחלה ואז ϕ לא צריך להתקיים כלל)
בינארי	$\phi R\psi$ או $\phi\ V\ \psi$	$\phi R\psi$ או $\phi\ V\ \psi$	**R**elease -	ϕ משחרר את ψ , ψ יתקיים לעולם או עד ש ϕ יתקיים איתו פעם אחת לפחות. (ψ חייב להתקיים לפחות פעם אחת אבל ϕ <u>לא</u> <u>חייב</u> להתקיים בסוף)

אונארי	$\square\ \phi$	$X\phi$	neXt	ϕ יתקיים בנקודת הזמן הבאה
אונארי	$<>\phi$	$F\phi$	**F**uture	ϕ יתקיים בנקודת זמן כלשהי בעתיד (חייב להתקיים בסופו של דבר)
אונארי	$\square\phi$	$G\phi$	**G**enerally	ϕ יתקיים תמיד.

הערה - בספרים שונים קיימים לעיתים סימונים שונים לקשרים טמפורליים אלו, למשל X מוחלף לעיתים עם N (מלשון next) וכן R מוחלף עם V . באלגוריתם המוצג בעבודה זו הקשר R מסומן ע״י V ולכן מעתה אתייחס אליו כ – V. כמו-כן חשוב להדגיש כי ה LTL-Linear Temporal Logic מתייחסת לציר זמן אחד, ציר ליניארי, ויש שפות לוגיות נוספות העוסקות במספר צירי זמן מסתעפים, עבורם קיימים קשרים נוספים אך הם אינם באים לידי ביטוי באלגוריתם הנ״ל ולכן לא ארחיב עליהם.

עבור כל נוסחא ב- LTL ניתן להגדיר את קבוצת המילים המקיימות את תנאי הנוסחא. כמובן שיש נוסחאות עבורם קבוצת המילים המתקיימות ריקה (למשל נוסחאות שמכילות סתירה לוגית בתוכן)

דוגמאות לנוסחאות בשפת ה- LTL:

כל הדוגמאות הן מעל השפה $D=\{a,b,p,q\}$

- $\xi = <> p$

פירוש: מתישהו p יופיע , ואח״כ לא משנה מה...

מילים המקיימות נוסחא זו:

aaaaabbbpaaabbb…

apbbbb….

paccccc….

- $\xi = <>\square p$

פירוש: מתישהו p יופיע תמיד

מילים המקיימות נוסחא זו:

abbpba…ppppppp…(p continue)

ppppppp….(p continue)

- $\xi = \Box(p \rightarrow \Box q)$

פירוש: תמיד אחרי p יופיע q (יכול להיות ש p לא יופיע כלל, ויכול להיות ש q יופיע לבד, אבל תמיד אחרי p יופיע q)

מילים המקיימות נוסחא זו:

abbaaaapqa….pq…abb…(after every p come q)
aabbqa…pq..(p can come alone)
ababba…..(no p appears)

- $\xi = \Box <> p$

פירוש: תמיד מתישהו יופיע עוד p

מילים המקיימות נוסחא זו:

abaa….pa….pb…ppab…p…(always there is another p)

- $\xi = p \cup q$

פירוש: q יופיע בסופו של דבר, ועד אז p "ישמור"

מילים המקיימות נוסחא זו:

pppppppppqab…….
pqab…..
qaab…..

- $\xi = pVq$

פירוש: p משחרר את q , ואם p לא מופיע אז q ימשיך לעד.

מילים המקיימות נוסחא זו:

qqqqq….
qqqpab…..
paba….

- $\xi = p \cup (t \cup m)$

פירוש: p יופיע קבוע עד ש t יופיע קבוע עד ש m יופיע (גם כאן אם t הופיע מיד אין צורך ב p, וכן אם m הופיע אין צורך ב t)

מילים המקיימות נוסחא זו:

pppptttmabbbab…
ttttttmabaa….
maba…..

2. אוטומט LGBA

2.1. הגדרת LGBA: Labeled Generalized Buchi Automaton

אוטומט בושי מתויג **A** הוא שישייה: $A=\langle Q, I, \to, F, D, L\rangle$ כאשר:

- Q היא קבוצה סופית של מצבים
- $I \subseteq Q$ הוא קבוצת מצבים תחיליים
- $\to$ היא פונקצית מעברים בין המצבים (מאיזה מצב לאיזה מצב יש מעבר)
- $F \subseteq Q$ קבוצת מצבים מקבלים
- D הוא קבוצת ה"אותיות" של האוטומט, האלפבית של האוטומט
- $L: Q \to 2^D$ הוא פונקצית תיוג, המגדירה לכל מצב אילו אותיות מתקיימות בו.

נשים לב כי באוטומט בושי "רגיל" פונקצית התיוג נעשית על המעברים, כלומר $L: Q \times D \to 2^Q$ ואילו באוטומט מתויג, התיוג נעשה על המצב, כלומר פונקצית התיוג מגדירה עבור כל מצב אילו אותיות הוא חייב להכיל.

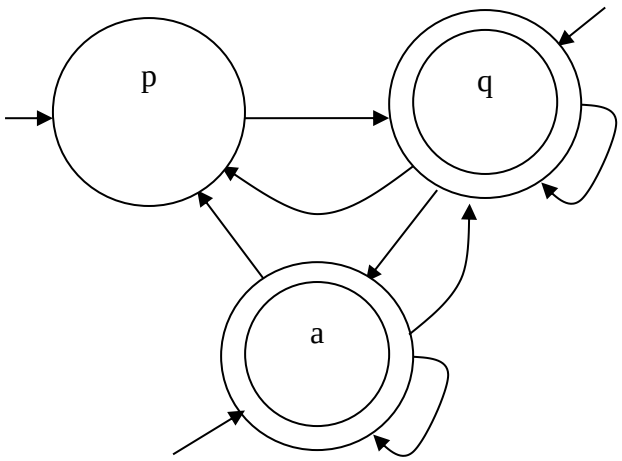
אוטומט בושי מקבל קלטים אינסופיים (רצף של אותיות מהתחום D), כאשר הרצה של מילה אינסופית באוטומט היא שבכל נקודה בה אנו נמצאים בקלט קיים מעבר למצב המכיל את האות הבאה בקלט.

נגדיר כי אוטומט "מקבל" קלט מסוים אם עבור הקלט קיימת ריצה על האוטומט שעוברת אינסוף פעמים במצב מקבל (מתוך קבוצה F) .

הערות:

- מצב באוטומט בו אין "הבטחה" לאות מסוימת יכול לקבל כל אות מהתחום.
- מצב באוטומט בו קיימת הבטחה ל $\neg p$ יכול לקבל כל אות מ-D חוץ מ p. דוגמא: עבור האלפבית $D=\{a,b,p,q\}$, $\neg p \equiv (a \vee b \vee q)$

2.2.דוגמאות לאוטומט LGBA



- עבור האלפבית: $D=\{a,p,q\}$

תרגום סמנטי: תמיד אחרי כל p מגיע q

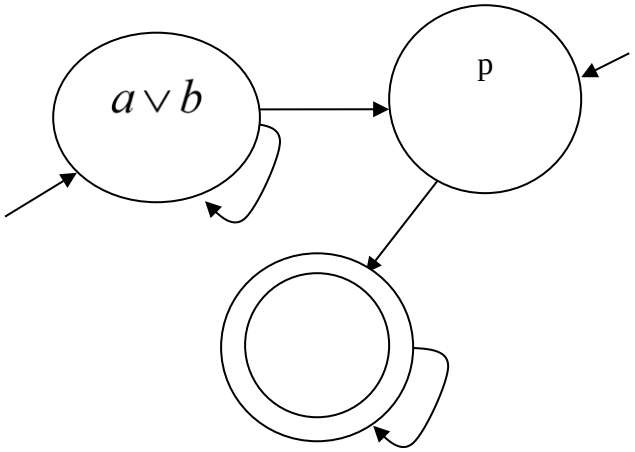
תרגום ל LTL: $\square(p\rightarrow\diamond q)$

מילים שהאוטומט מקבל:

aqaapqpqpqpqpqqqq….

pqpqpqqqqaaaa…

qqqaqqaqqaaa……



- עבור האלפבית: $D=\{a,b,p\}$

תרגום האוטומט: מתישהו יגיע p

תרגום ל LTL: $\diamond p$

מילים שהאוטומט מקבל:

abbbbbbpppppaa…….

abbbaaabbpabbb…

paab…..

חשוב לשים לב כי ניתן למצוא ריצה גם למילים .....aaaaaaaa או ....abababababa שלא מכילות את p אולם ריצות אלו אינן ריצות "מקבלות" – accepting executions מאחר ולא מתקיים התנאי שהריצה עוברת אינסוף פעמים במצב "מקבל".

3. שיטת ה- Model Cheking

3.1.חשיבות אימות התוכנה

בימינו, יותר ויותר מערכות הופכות להיות ממוחשבות. כל עולם המחשבים, התעשייה, התחבורה, הצבא, הפיננסים, המדע מבוסס על מערכות ממוחשבות. במהלך העשורים האחרונים, פעמים רבות קרה שארעה תקלה קריטית/הרסנית במערכת ממוחשבת, למרות כל הבדיקות שנעשו על המערכת. לעיתים התקלה גורמת להרס של שנות בניה ומחקר רבות, ועלות התקלה יכולה להגיע למיליונים רבים.

להלן מספר מצומצם מאוד של דוגמאות למקרים בהם כשל קריטי במערכת ממוחשבת גרם לנזק עצום:

- בשנת 1990 נפלה רשת הטלפוניה של AT&T למשך 9 שעות וגרמה להפסד של מאות מיליוני דולרים, וזאת עקב שגיאת תוכנה שנגרמה עקב פירוש שגוי של פקודת BREAK בשפת ה-C.

- בשנת 1994 נתגלתה שגיאה במעבדי פנטיום של חברת אינטל, כאשר חלק מפעולות חילוק נקודה צפה נתנו תוצאה שגויה (1 מתוך 9000000 פעולות נתנה תוצאה שגויה), שבעטיה נאלצה החברה להחליף את כל המעבדים, מה שגרם לה להפסדים של חצי מיליארד דולר. התקלה נגרמה עקב שגיאה במימוש אלגוריתם לחלוקה בייצוג נקודה צפה.

- בשנת 1996 התרסק הטיל ARIANE5 עקב שגיאה בתכנת הבקרה של הטיל, שנגרמה עקב המרה שגויה מ floating point ל- signed integer. טעות זו גרמה להספד של 500 מליון יורו.

כאשר יותר ויותר מערכות הופכות להיות תלויות תוכנה, וכאשר העלויות שלהן והרגישות שלהן גדלות יותר ויותר, אמינות המערכת כולה תלויה בתוכנה ושגיאות עלולות להיות הרסניות ויקרות, כמובן במערכות רגישות: רפואיות, צבאיות, בקרת תחבורה ותקשורת אך גם הן במערכות קטנות המיוצרות בייצור המוני. ולכן, קיימת חשיבות ראשונה במעלה לאמת את התקפות המערכת הממוחשבת, כלומר לבדוק שהמערכת מקיימת את כל הדרישות שהוגדרו לה.

קיימות מספר שיטות עיקריות לאימות תוכנה כגון: סקירת קוד ידנית, בדיקות דינאמיות בהרצת הקוד וכמובן גם בדיקות מודל בהם עבודה זו עוסקת, כאשר לכל אחת מהשיטות יש יתרונות וחסרונות, ובחירת השיטה תלויה בהתאמתה למערכת הנבדקת. בסעיף הבא אפרט יותר על שיטת "בדיקות המודל", ה- "model checking".

3.2.בדיקות המודל – model checking

כפי שטענתי בסעיף הקודם, החשיבות של אימות המערכות עלתה בשנים האחרונות. אחת מהשיטות העיקריות לאימות תוכנה ומערכות ממוחשבות היא שיטת בדיקות המודל. במאמר

מוסגר אזכיר כי בשנים האחרונות זכו בפרס טיורינג המכובד הניתן על הישג יוצא דופן בתחום מדעי המחשב, מספר חוקרים על מחקר שתרם לפיתוח אימות התוכנה בשיטת בדיקות המודל, ראשון להם היה פרופ׳ אמיר פנואלי,שזכה בפרס בשנת 1996 ״על הכנסת לוגיקת הזמן לתחום מדעי המחשב ועל תרומה רבה לתחום אימות תוכנה ומערכות״ (מתוך הנימוקים לפרס טיורינג, אתר אגודת ACM) ועוד בשנת 2008 זכות 3 חוקרים: אדמונד קלארק, אלן אמרסון וג׳וזף סיפאקיס בפרס טיורינג ״על תפקידם בפיתוח שיטת בדיקות מודל כטכנולוגיית אימות יעילה שאומצה באופן נרחב בתעשיות התוכנה והחומרה״ (מתוך הנימוקים לפרס, אתר אגודת ACM).

כאשר אנו רוצים לאמת תקפות של מערכת קיימת, או לחילופין לבנות מערכת חדשה, אנו צריכים להגדיר מהן הדרישות שאנו רוצים שהמערכת תעמוד בהן. לכל מערכת אוטומטית יש את דרישות הקיום שלה, החל ממערכת רמזורים, דרך ניהול מדפסת או כונן משותף, רשת תקשורת ארצית ועד לבניית מערכת פיקוד ממוחשבת אוטומטית למטוס. קיימות מספר תכונות פרקטיות שכמעט כל מערכת אוטומטית צריכה לעמוד בהן על-מנת להיחשב כמערכת תקפה. בין התכונות הבסיסיות נוכל למצוא את:

Fairness – הגינות: כל מערכת תקינה בה רצים מספר תהליכים חייבת לקיים את תכונת ההגינות, כלומר שאין תהליך שמורעב, כלומר שנמנעים ממנו משאבי המערכת. מניעת ההרעבה היא ע״י הדרישה שכל תהליך שביקש משאב מסויים בסופו של דבר יקבל אותו, תמיד.

דרישת ההגינות בתרגום לשפת ה- LTL: $GFp_1 \rightarrow GFp_2$

Liveness – חיות: כל מערכת צריכה לדאוג לכך שלא נוצרת ״תקיעה״ בה מספר תהליכים חוסמים זה את זה מלהתקדם, נעילה המכונה בשם ״deadlock״ ובכל נקודת זמן, כל תהליך יתקדם בסופו של דבר. דרישת החיות בתרגום לשפת ה- LTL:

$G(p_1 \rightarrow Fp_2)$

Mutual exclusion - המניעה ההדדית: כל מערכת מקבילית בה יש משאבים קריטיים כגון מאגר נתונים משותף, צריכה לדאוג שלא יקרה מצב בו שני תהליכים מקבילים ישתמשו באותו משאב בו-זמנית, דבר העלול לגרום לתקלות ברורות. דרישת המניעה ההדדית בתרגום לשפת ה- LTL: $G(\neg(crit_{p_1} \wedge crit_{p_2}))$

האלגוריתם בו אדון בפרק הבא בודק הלכה למעשה נוסחאות בלוגיקה טמפורלית ובונה מודל של אוטומט LGBA עבור הנוסחא הנבדקת.

4. האלגוריתם של ורדי-פלד-גרט-וולפר.

בשנת 1995 הציגו ורדי, פלד, וולפר וגרט את האלגוריתם המוצג בעבודה זו, שהינו פיתוח של אלגוריתמים קודמים להעברת נוסחא ב LTL לאוטומט בושי שפותחו לראשונה ע״י ורדי, וולפר

וסיסטלה בשנת 1983. האלגוריתם זה יעיל יותר באופן משמעותי מקודמיו. בפרק זה אציג תחילה הגדרות בסיסיות החשובות להבנת האלגוריתם, אציג את מבנה הנתונים בו משתמש האלגוריתם, ולבסוף אבאר את השיטה בה בונה האלגוריתם את האוטומט שהוא התוצר הסופי של האלגוריתם. (האלגוריתם מצורף בנספח 1)

4.1. הגדרות

- **תרגום של האופרטורים הטמפורליים**:

מחברי האלגוריתם הנ"ל רצו ליצור אלגוריתם פשוט ככל שניתן, שיהיה נוח גם להבנה, ובעיקר להוכחה, וכתוצאה מכך הם הכניסו לאלגוריתם את היכולת לעבד רק נוסחאות המכילות את האופרטורים: $(\neg, \vee, \wedge, V, \bigcup)$ מתוך הנחה שכל נוסחא ב LTL ניתן לתרגם לפני הרצתה על האלגוריתם לנוסחא המכילה את האופרטורים הנ"ל. כשם שבלוגיקה הקלאסית ניתן לתרגם כל נוסחא לנוסחא המכילה רק את האופרטורים $\neg, \vee, \wedge$ (ואף ניתן להסתפק בקבוצות $(\neg, \vee)$ או $(\neg, \wedge)$ שהינן "קבוצות קשרים מספיקות") כך גם בלוגיקה הטמפורלית ניתן לתרגם כל אחד מהאופרטורים שהצגתי לעיל לנוסחא המכילה רק את האופרטורים $(\neg, \vee, \wedge, V, \bigcup)$ המעובדים באלגוריתם הנ"ל. בטבלה שלהלן אציג תרגום אפשרי (לעיתים ניתן להציג מספר תרגומים שקולים) לכל אחד מהאופרטורים שאינם מעובדים באלגוריתם:

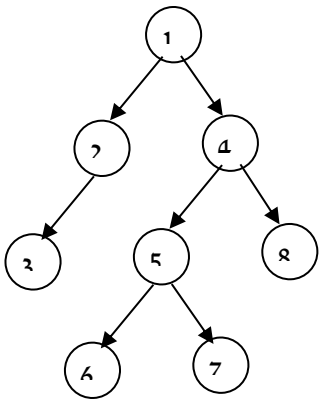
האופרטור	התרגום הרקורסיבי	הערות
$<>\phi$	$TU\phi$	הנוסחא נכונה עד ש ϕ יתקיים (ϕ חייב להתקיים בסופו של דבר)
$\Box\phi$	$FV\phi$	F מייצג FALSE כלומר ברגע שהוא יתקיים, כלומר ישחרר את ϕ, הנוסחא "נופלת"

ההתייחסות לאופרטור: $\Box\varphi$ שונה במקצת מכיוון שהאלגוריתם עצמו מעבד כל נוסחא לנתונים שחייבים להתקיים היום ומהם ההתחייבויות למחר, ולכן ההתייחסות לנוסחא $\Box\varphi$ תהיה בכך שהיום לא צריך להתקיים כלום, כלומר TRUE בכל מצב, ומחר צריך להתקיים φ.

- **שיטת DFS**

האלגוריתם מתקדם בצורה רקורסיבית בשיטת הסריקה DFS: Depth First Search כלומר הוא קודם יורד עד לשורש הנוסחא (הכי עמוק שאפשר) ורק אחר כך מעבד את הצמתים שברוחב. ניתן לסרוק נתונים בשיטת DFS ימני או DFS שמאלי, כלומר בכל

פיצול בעץ נתקדם קודם לצד קבוע ורק לאחר שנמצה את הענף שבחרנו נחזור נעבור לצד השני. להלן דוגמא לסדר ביקור בצמתים בעץ בסריקת DFS שמאלי



4.2. מבנה הנתונים של האלגוריתם

האלגוריתם משתמש בשני מבני נתונים עיקריים לצורך בניית גרף האוטומט, משתנה מסוג רשומה (record) עם מספר שדות, ורשימה (list):

רשומה node המכילה את השדות הבאים:

- **Name**: שם הצומת – שם ייחודי לכל צומת.
- **Incoming**: שמות כל הצמתים שלהם יש מעבר את הצומת הנוכחי. צומת שמהווה מצב תחילי, שדה ה - incoming שלה יכיל את הביטוי "init".
- **New**: קבוצת כל הנוסחאות הצריכות להיות ממומשות בצומת זו ועדיין לא עובדו.
- **Old**: קבוצת כל הנוסחאות הצריכות להיות ממומשות בצומת זו וכבר עובדו.
- **Next**: קבוצת הנוסחאות שצריכות להתממש בצמתים שאליהם יוביל הצומת הנוכחי

בנוסף, קיים באלגוריתם עוד שדה אחד במחלקה זו: **father** , אך שימושו הוא רק להוכחת נכונות האלגוריתם ואין בו שימוש בבניית האוטומט.

ורשימה **Nodes_Set** המכילה את שמות כל הצמתים שעיבודם הסתיים ונכנסים לאוטומט. לא כל הצמתים נכנסים לרשימה זו, אלא רק אלו השונים זה מזה ושאינם מכילים סתירה פנימית. ארחיב על כך בהמשך.

בסיום ריצת האלגוריתם נקבל כפלט את רשימת הצמתים שמרכיבים את האוטומט, אולם לפני בניית האוטומט עדיין צריך לבחור את קבוצת המצבים המקבלים. התנאי להפוך להיות מצב

״מקבל״ הוא שאין בשדה הnext של השדה ״הבטחות״ שעדיין לא מומשו או שלא ממומשות בצומת הנ״ל.

4.3. האלגוריתם

הערה: בסוגריים [ ] אציין את השורות המתאימות בקוד האלגוריתם

מטרת האלגוריתם הנ״ל היא לבנות מודל של אוטומט שיקבל את כל הצירופים שמספקים נוסחא טמפורלית נתונה. במידה והנוסחא לא ספיקה (unsatisfiable) בניית האוטומט תגיע לסתירה ובכך תהיה הוכחה שהנוסחא מכילה סתירה.

האלגוריתם מתבסס על שיטת ה- tableau construction כלומר שכל נוסחא/תת-נוסחא המכילה אופרטור לוגי או טמפורלי מתורגמת לנוסחת ״או״ או ״וגם״ , בהתאם לאופרטור המעובד **[שורות 14-31]**. כאשר התרגום הוא לנוסחת ״או״ : $\varphi \vee \psi$, כל תת נוסחא φ ו- ψ תיצור ״node״ חדש, אחד שחייב לקיים את φ ואחד שחייב לקיים את ψ. כאשר התרגום הוא לנוסחת ״וגם״ $\varphi \wedge \psi$ האלגוריתם יצור ״node״ חדש אחד בלבד שחייב לקיים גם את ψ וגם את φ.

האלגוריתם משתמש הפונקציות: $new1(\eta)$, $next1(\eta)$, $new2(\eta)$ המתוארות בטבלה להלן, הקובעות עבור כל אחד מהאופרטורים V , $\bigcup$, $\vee$ מה חייב להתקיים ״היום״ בכל אחד מהצמתים החדשים שנוצרים (נכנס לשדה ה new ברשומת הצומת החדשה) ומה חייב להתקיים ״מחר״ (נכנס לשדה ה next ברשומת הצומת הראשונה בלבד)

טבלת $new1(\eta)$, $next1(\eta)$, $new2(\eta)$:

η	$New1(\eta)$	$Next1(\eta)$	$New2(\eta)$
$\mu \bigcup \psi$	$\{\mu\}$	$\{\mu \bigcup \psi\}$	$\{\psi\}$
$\mu V \psi$	$\{\psi\}$	$\{\mu V \psi\}$	$\{\mu, \psi\}$
$\mu \vee \psi$	$\{\mu\}$	ϕ	$\{\psi\}$

האלגוריתם ימשיך לעבד את הצמתים החדשים באופן רקורסיבי, על פי סדר סריקת ה DFS עד שהנוסחא תכיל ליטרלים בלבד שחייבים להתקיים היום ב״node״ המעובד ונוסחאות שהן הבטחות שצריכות להתקיים ״מחר״. אז נסמן אותה ב- כ״צומת מעובדת״ (בדוגמאות להלן מסומן ב- □) ונבחן את מה שצריך להיות ממומש מחר **[שורות 8-10]**.

[שורות 5-7] במידה ובעיבוד נוסחא מסוימת מגיעים לנוסחא מעובדת הזהה בהבטחות להיום ובהבטחות למחר (שדות ה old וה next) ל״צומת מעובדת״ הנמצאת כבר ברשימת ה **nodes_set**

(בדוגמאות להלן מסומן ב $\times$), נסיים את הטיפול בנוסחא זו ונתייחס לכך בסימון המעברים בין הצמתים באוטומט העתידי כפי שאפרט בהמשך.

"**עולם שכולו טוב**": נקודה חשובה נוספת בהבנת האלגוריתם היא שמכיוון שהאלגוריתם בונה אוטומט בושי, המקבל קלט אינסופי, גם לאחר סיפוק כל ההתחייבויות של הנוסחא האלגוריתם ממשיך ליצור צומת נוספת, בה כבר אין הבטחות, (אני מכנה צומת זו: "עולם שכולו טוב") המכילה גם מעבר מעגלי לתוך עצמה, והיא כמובן מצב "מקבל", כך שהקלט האינסופי ימשיך לרוץ באוטומט כל מקרה, וגם ויעבור אינסוף פעמים במצב "מקבל" כפי שמתבקש בקלט מקובל באוטומט בושי. כאמור, רק קלט שמילא את כל התחייבויותיו יוכל להגיע למצב זה.

סימון מצבים מקבלים באוטומט: רק הצמתים שאינם מכילים "הבטחות למחר" או שמכילים "הבטחות למחר" שממומשות בהמשך האוטומט יסומנו כמצבים מקבלים באוטומט העתידי.

סימון המעברים בין הצמתים: הנוסחא הראשונית וכל תתי הנוסחאות הנוצרים ממנה כתוצאה מעיבוד אופרטורים, מסומנים כמצבים תחיליים. בכל פעם שמסיימים לעבד נוסחא והיא מסומנת כ"צומת מעובדת" וממשיכים לבחון את ההבטחות למחר, כל העיבודים החדשים והצמתים שיווצרו מהם יהיו ה"בנים" של הצומת המעובדת ממנה נלקחה ה"הבטחה למחר", כלומר יהיו מעברים מצומת ה"אבא" לכל התולדות שנוצרו מההבטחה למחר. בכל פעם שמסיימים לעבד נוסחא ומגלים שכבר קיימת צומת מעובד זהה לזו **[שורות 7-5]**, הצומת הקיימת תקבל מעבר גם מה"אבא" של הצומת ה"מיותרת" שעובדה זה עתה.

בפרק הבא אדגים מספר ריצות של האלגוריתם על נוסחאות טמפורליות, פשוטות ומורכבות, אפרט את עיבוד הנוסחא עפ"י שיטת הטבלו, אצור את האוטומט המתקבל ע"י הרצת האלגוריתם בליווי הסברים.

5. דוגמאות להרצת נוסחאות LTL באלגוריתם ובניית אוטומט עבורן

- **הרצת הנוסחא** $p \cup t$

$p \cup t$

$\boxed{p, \Box(p \cup t)}_1$ $\quad$ $\not{X}_6$

$\downarrow$

$p \cup t$

$\not{p, \Box(p \cup t)}_2$ $\quad$ $\boxed{t}_3$

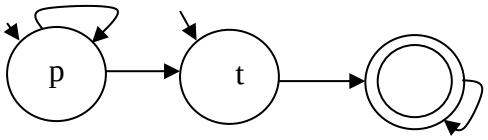
$\downarrow$

$\boxed{\varnothing}_4$

$\downarrow$

$\not{\otimes}_5$

האוטומט הנוצר:



- **הרצת הנוסחא** $\Box\Diamond p$

$\Box\Diamond p$

$F \; V \; \Diamond p$

$F \; V \; (T \cup p)$

$\{(T \cup p) \wedge \Box[F \; V \; (T \cup p)]\} \vee [F \wedge (T \cup p)]$

$(T \cup p) \wedge \Box[F \; V \; (T \cup p)]$ $\quad$ $\not{F \wedge (T \cup p)}_8$

$T \cup p, \Box[F \; V \; (T \cup p)]$ $\quad$ *false* $\otimes$

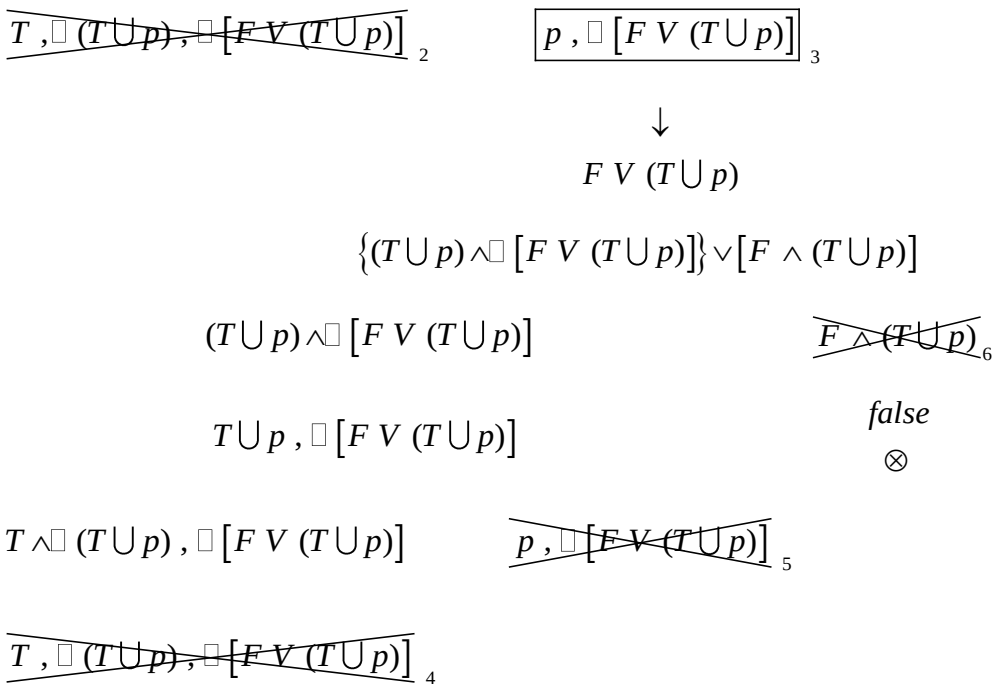
$T \wedge \bigcirc(T \cup p), \Box[F \; V \; (T \cup p)]$ $\quad$ $\not{p, \Box[F \; V \; (T \cup p)]}_7$

$\boxed{T, \bigcirc(T \cup p), \Box[F \; V \; (T \cup p)]}_1$

$\downarrow$

$T \cup p, F \; V \; (T \cup p)$

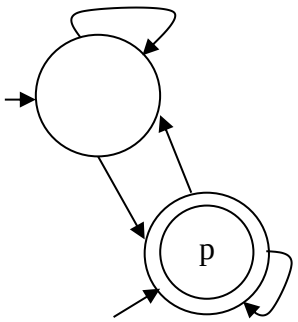
$T \cup p, \not{T \cup p}, \Box[F \; V \; (T \cup p)]$



הסבר:

- $\square_1$ היא הצומת המעובדת הראשונה, מצב תחילי, חייב להתקיים p וקיימות הבטחות למחר
- $\square_2$ היא בן של $\square_1$ ולכן מכילה מעבר ממנו, אך היא זהה לו גם בהבטחות להיום וגם בהבטחות למחר, ולכן $\square_1$ יכיל מעבר לעצמו.
- $\square_3$ הצומת המעובדת השנייה, תקבל מעבר מ $\square_1$, חייב להתקיים p וקיימת הבטחה למחר.
- $\square_4$ היא בן של $\square_3$ וגם זהה ל $\square_2$, ולכן $\square_3$ יכיל מעבר ל $\square_2$.
- $\square_5$ היא בן של $\square_3$ אך גם זהה לו ולכן $\square_3$ יכיל מעבר לעצמו.ממנו
- $\square_6$ היא צומת שמכילה F ולכן נפסלת.
- $\square_7$ דומה ל $\square_3$, והוא מצב תחילי (הבן של הנוסחא המקורית) ולכן $\square_3$ יהיה מצב תחילי.
- $\square_8$ היא צומת שמכילה F ולכן נפסלת.

האוטומט שנוצר:



הערה: חשוב לשים לב כי $\square_3$ הינו מצב מקבל, **למרות** שקיימות בו עדיין הבטחות לעתיד, כפי שניתן לראות בפירוט הרצת האלגוריתם, מכיוון שהאפשרות שקלט יהיה קלט מקבל תתקיים אך ורק אם הוא יעבור אינסוף פעמים בצומת זו, כלומר שתמיד קיים עוד p .

- **הרצת הנוסחא** $\Box(p \cup t)$

$$\Box(p \cup t)$$

$$p \cup t \,,\ \Box\left[F \vee (p \cup t)\right]$$

$$\boxed{p \,,\ \Box(p \cup t) \,,\ \Box\left[F \vee (p \cup t)\right]}_{1} \qquad \cancel{t \,,\ \Box\left[F \vee (p \cup t)\right]}_{6}$$

$$\downarrow$$

$$p \cup t \,,\ F \vee (p \cup t)$$

$$\cancel{p \cup t} \,,\ p \cup t, \Box\left[F \vee (p \cup t)\right] \qquad p \cup t \,,\ F \,,\ (p \cup t)$$

$$\cancel{p \,,\ \Box(p \cup t) \,,\ \Box\left[F \vee (p \cup t)\right]}_{2} \qquad \boxed{t \,,\ \Box\left[F \vee (p \cup t)\right]}_{3} \qquad \textit{contain false}$$

$$\downarrow \qquad\qquad \otimes$$

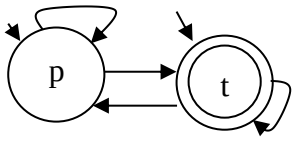
$$F \vee (p \cup t)$$

$$p \cup t, \Box\left[F \vee (p \cup t)\right] \qquad p \cup t \,,\ F \,,\ (p \cup t)$$

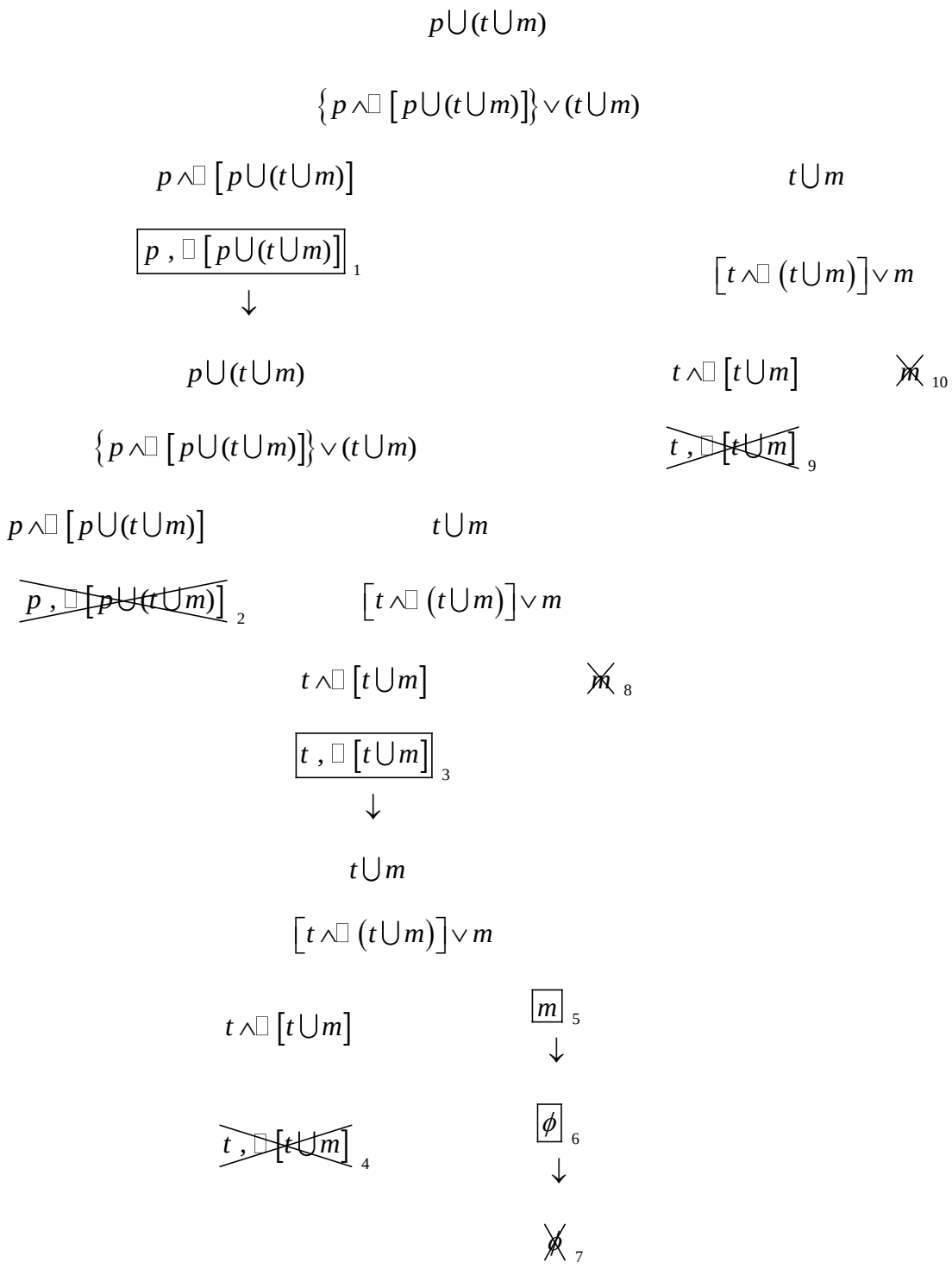
$$\cancel{p \,,\ \Box(p \cup t) \,,\ \Box\left[F \vee (p \cup t)\right]}_{4} \qquad \cancel{t \,,\ \Box\left[F \vee (p \cup t)\right]}_{5} \qquad \textit{contain false}$$

$$\otimes$$

האוטומט שנוצר:



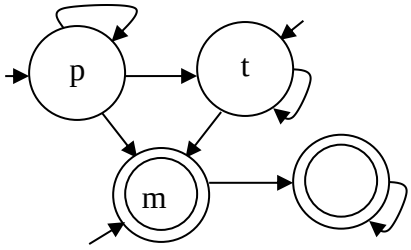
- **הרצת הנוסחא** $p \cup (t \cup m)$



הסבר:

- $\square_1$ היא הצומת המעובדת הראשונה, מצב תחילי, חייב להתקיים p וקיימת הבטחה למחר
- $\square_2$ היא בן של $\square_1$ ולכן מכילה מעבר ממנו, אך היא זהה לו גם בהבטחות להיום וגם בהבטחות למחר, ולכן $\square_1$ יכיל מעבר לעצמו.
- $\square_3$ הצומת המעובדת השנייה, תקבל מעבר מ $\square_1$, חייב להתקיים t וקיימת הבטחה למחר.
- $\square_4$ היא בן של $\square_3$ וגם זהה לו, ולכן $\square_3$ יכיל מעבר לעצמו.
- $\square_5$ הצומת המעובדת השלישית. הבן של $\square_3$ ולכן יכיל מעבר ממנו. $\square_5$ הוא המצב הראשון שאינו מכיל הבטחות למחר, לכן הוא יהיה מצב "מקבל", וממנו יוביל מעבר לצומת ללא הבטחות, "עולם שכולו טוב" $\square_6$ שגם הוא יהיה מצב מקבל ויכיל מעבר לעצמו (בגלל שהבן שלו, $\square_7$ זהה לו).
- $\square_8$ הבן של $\square_1$ וזהה ל $\square_5$ ולכן יהיה מעבר מ $\square_1$ ל- $\square_5$.
- $\square_9$ דומה ל $\square_3$, והוא מצב תחילי (הבן של הנוסחא המקורית) ולכן $\square_3$ יהיה מצב תחילי.
- $\square_{10}$ דומה ל $\square_5$ והוא מצב תחילי , ולכן $\square_5$ יהיה מצב תחילי.

האוטומט שנוצר:



6. הוספות ושיפורים לאלגוריתם

במהלך השנים הוצעו לאלגוריתם זה כמה וכמה גרסאות משופרות, הן בכדי לייעל את בניית האוטומט והן בכדי להוסיף לו תכונות נוספות. בסעיף זה אסקור מספר הצעות לשיפור, חלקם הוצע ע"י המחברים עצמם, וחלקם אני מציע להוסיף ע"מ להרחיב את צורת הנוסחאות ב-LTL איתם האוטומט יודע להתמודד מבלי הצורך בתרגום מקדים.

6.1. הצעות המחברים

אחת ההצעות הראשונות שהוצעה ע"י המחברים עצמם הייתה להוסיף לאלגוריתם את היכולת להתמודד עם נוסחא המכילה אופרטור "next". כפי שהזכרתי לעיל, שיטת העיבוד של האלגוריתם היא לחלק בין התנאי שצריכים להתקיים היום ובין התנאים שצריכים להתקיים מחר, ולכן, ההצעה הפשוטה וההגיונית, היא להוסיף בשלב ה *case* η *of* (המופיע בשורה 14) אופציה נוספת עבור האופרטור $\bigcirc\mu$, היוצרת node חדש כאשר שדה ה new שלו נשאר כפי שהיה קודם, ואילו שדה ה next יקבל את מה שהיה ב node הקודם בתוספת μ. קטע הקוד (לקוח מתוך המאמר) שיתווסף הוא כדלקמן:

$\eta = X\,\mu \Rightarrow$
return(expand([*Name* $\Leftarrow$ *Name(Node)*, *Father* $\Leftarrow$ *Father(Node)*,
Incoming $\Leftarrow$ *Incoming(Node)*, *New* $\Leftarrow$ *New(Node)*, *Old* $\Leftarrow$ *Old(Node)* $\cup\{\eta\}$,
Next=Next(Node) $\cup\{\eta\}$], *Nodes_Set*))

הוספה זו לא נכללה באלגוריתם המקורי אך נכתבה בצמוד לו, לטענת הכותבים, בכדי להשאיר את האלגוריתם פשוט להוכחה אך הוספה זו כמובן אינה פוגמת בנכונותו. הוספה זו מאפשרת לטפל גם בנוסחאות המכילות הבטחות למחר באופן ישיר.

הצעה נוספת של המחברים נועדה לייעל את עבודת האלגוריתם ולחסוך במשתנים חדשים ובזיכרון. ההצעה היא שבכל פעם שיש צורך לפצל צומת לשני צמתים חדשים, במקום ליצור שני צמתים חדשים, ניצור צומת אחת חדשה בלבד, ונעדכן את הצומת הקיימת עפ"י הנתונים שצריכים להיות בצומת השנייה, מכיוון שתפקידה של הצומת הנוכחית מסתיים לאחר יצירת הצמתים ה"בנים".

6.2. ההצעות שלי

הוספת האופרטורים $\bigcirc$, $\Diamond$

מכיוון שלנוסחאות המכילות את האופרטורים $\bigcirc$, $\Diamond$ יש נוסחאות שקולות בעזרת האופרטורים $\vee$, V, $\bigcup$ הקיימים באלגוריתם, ניתן להוסיף לאלגוריתם את היכולת להתמודד עם הנוסחאות $\Diamond\mu$, $\bigcirc\mu$ ע"י הרחבת טבלת הפונקציות $New2(\eta)$, $Next1(\eta)$, $New1(\eta)$ גם לאופרטורים אלו.

נוסחא שקולה ל μ היא: $T \cup \mu$ (T משמעו TRUE) כלומר הנוסחא נכונה ללא תנאים בתנאי ש μ יתקיים מתישהו בעתיד, ונוסחא שקולה ל $\Box\mu$ היא: $F V \mu$, כלומר μ חייב להתקיים כל הזמן, וברגע שהוא ישתחרר, זה יקרה ע"י F (F משמעו FALSE) כלומר הנוסחא תיפול.

הטבלה המורחבת תיראה כך:

η	$New1(\eta)$	$Next1(\eta)$	$New2(\eta)$
$\mu \cup \psi$	$\{\mu\}$	$\{\mu \cup \psi\}$	$\{\psi\}$
$\mu V \psi$	$\{\psi\}$	$\{\mu V \psi\}$	$\{\mu, \psi\}$
$\mu \vee \psi$	$\{\mu\}$	$\varnothing$	$\{\psi\}$
$\Diamond \mu$	$\varnothing$	$\{T \cup \mu\}$	$\{\mu\}$
$\Box \mu$	$\{\mu\}$	$\{F\ V \mu\}$	$\{F, \mu\}$

לאחר הרחבת הטבלה, אין צורך להוסיף פקודות נוספות באלגוריתם, והעיבוד עבור האופרטורים החדשים ייעשה אף הוא בשורות 27-21. השינוי היחיד יהיה בשורה 20 בה נוסיף את האופציות החדשות כדלהלן:

20 $\eta = \mu \cup \psi \ \ or \ \ \mu V \psi \ \ or \ \ \mu \vee \psi \ \ or \ \Box \mu \ \ or \ \Diamond \mu$

איחוד הטיפול באופרטורים $\Box$, $\Diamond$, $\wedge$

עפ"י הטבלה לעיל, פיצול האופרטור $\Box\mu$ מכיל סתירה ב $New2(\eta)$ (מכיוון שהוא מכיל בתוכו את False) ולכן אפשר לוותר על יצירת $New2$ וכך לחסוך זמן ריצה וזיכרון, וליצור רק אך צומת חדשה אחת. לשם כך נצטרך להוציא אותו מה case הזה ולתת לו case מיוחד כמו שעשינו לאופרטור $\Box\mu$ בסעיף הקודם.

למעשה אפשר לשים לב ששלושת האופרטורים: $\Box$, $\Diamond$, $\wedge$ מעובדים בצורה זהה בכך שעיבודם נעשה על ידי צומת אחת בלבד (לא חדשה, אלא עדכון של הצומת המעובדת), ונבדלים זה מזה רק בערכי ה- new וה next של צומת זו. לכן, ניתן להוסיף לטבלה לעיל עבור כל אחד משלושת האופרטורים, איזה ערך יתקבל מהפונקציה $New1(\eta)$ ואיזה ערך יתקבל מהפונקציה $Next1(\eta)$ כדלהלן:

η	$New1(\eta)$	$Next1(\eta)$
$\mu \wedge \psi$	$\{\mu, \psi\}$	$\varnothing$
$\neg\mu$	$\varnothing$	$\{\mu\}$
$\neg\mu$	$\{\mu\}$	$\{F\ V\mu\}$

כעת, נשנה את שורות 28-31 כך שיכילו את הטיפול בשלושת האופרטורים באופן הבא:

```
28   η = Xμ  or  μ∧ψ  or  Gμ ⇒
29      return(expand([Name⇐Name(Node), Father⇐Father(Node),
30           Incoming⇐Incoming(Node), New⇐New(Node)∪({New1(η)}\Old(Node)),
31           Old⇐Old(Node)∪{η}, Next=Next(Node)∪{Next1(η)}], Nodes_Set))
```

שינוי תנאי הזהות בשדות old ו next לשיפור יעילות האלגוריתם

במהלך חקירתי את האלגוריתם, הרצתי על האלגוריתם נוסחאות רבות ב-LTL וקיבלתי אוטומטים סופיים המהווים מודל לאותה נוסחא מורצת. במספר הרצות קיבלתי כתשובה אוטומט סופי שהיה חוקי ותקין, אך הכיל מספר גבוה של צמתים מכפי שציפיתי. בבדיקה חוזרת ונשנית של ההרצות זיהיתי כי תנאי הבדיקה בשורה 5 באלגוריתם גורם במקרים מסוימים לניפוח האוטומט המתקבל. לאחר התייעצות עם המנחה שלי הגענו למסקנה שאכן שיפור קל באלגוריתם יחסוך במקרים רבים פעולות רקורסיביות ארוכות שרק מנפחות את האלגוריתם מבלי להוסיף לו אלמנטים מהותיים, וזאת מבלי לגרוע בנכונות האלגוריתם.

השינוי המוצע:

בשורה 5 במקום:

if $\exists ND \in Nodes\_Set$ with $Old(ND)=Old(Node)$ and $Next(ND)=Next(Node)$

יש לכתוב:

if $\exists ND \in Nodes\_Set$ with $Old(ND) \subseteq Old(Node)$ and $Next(ND)=Next(Node)$

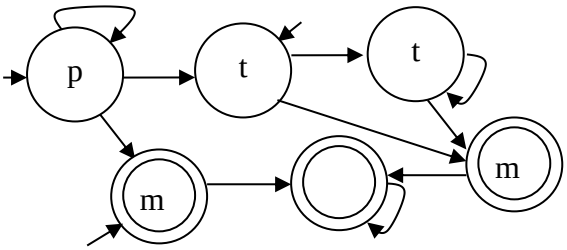
הסבר:

בשורות 3-5 האלגוריתם בודק את ה node שהתקבל כקלט בפונקצית expend : אם שדה ה-new שלו ריק (כלומר אין יותר הבטחות חדשות שיש לממש) בודקים האם קיים כבר ב nodes_set שנבנה עד כה node המכיל שדות old ו- next שזהים לחלוטין (שוויון =) שדות המקבילים להם ב node החדש. אם כן, אין צורך להוסיף node זה ל- nodes_set אלא רק לקחת ממנו מידע לגבי שדה ה-incoming שלו.

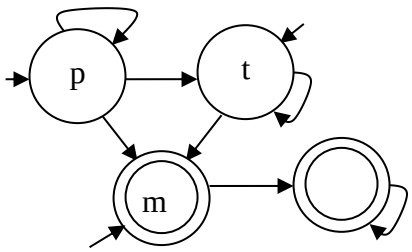
בשלב זה, במידה ושדה ה old של ה node החדש מוכל בשדה ה-old של ה- node הקיים לא יתקיים שוויון, וגם node זה יצורף ל- node_set מבלי שהוא מכיל הבטחות יתרות על אלו שקיימות ב-node הישן, וזה גורם לניפוח האוטומט המתקבל ולהמשך קריאות רקורסיביות מיותרות. אחרי שינוי השורה, תנאי הקיום יתקיים וה node החדש רק יוסיף את ערכי ה incoming שלו אל ה node הקיים.בהצעות לשיפור המופיעות בסוף המאמר, מתייחסים כותבי המאמר לעובדה כי לא כל נוסחא המעובדת צריכה לעבור אוטומטית לשדה ה – old , ואכן אם במהלך בניית האוטומט לא כל תת-נוסחא תצטרף לשדה ה- old אלא רק האטומים והליטרלים, זה ימנע מהאוטומט להתנפח שלא לצורך.

דוגמא ליעילות השיפור המוצע :

עבור הרצת הנוסחא : $p \cup (t \cup m)$ באלגוריתם כמו שהוא יתקבל האוטומט הבא :



עבור הרצת הנוסחא : $p \cup (t \cup m)$ באלגוריתם אחרי השינוי המוצע יתקבל האוטומט הבא :



7. סיכום

בעבודה זו חקרתי מאמר שנכתב בשנת 1995 ע״י 4 חוקרים, אשר פיתחו אלגוריתם נוח ופשוט לאימות של נוסחאות טמפורליות ובניית מודל אוטומט עבורן. כשקיבלתי לידיי את המאמר והאלגוריתם, הכרתי קצת את עולם האוטומטים ולוגיקה הקלאסית והטמפורלית, אולם כשנכנסתי לעובי הקורה הבנתי כי עליי לעשות כברת דרך על מנת שהידע הקודם שלי יגיע לנדרש במאמר. רק לאחר שרכשתי את הידע על האוטומטים המיוחדים אותם יוצר האלגוריתם, והכרתי לעומק את כל האופרטורים הטמפורליים הנוספים, בהם משתמש האלגוריתם, התחלתי להריץ נוסחאות וליצור מהם אוטומטים.

בעזרת מנחה העבודה, ברוגע, במיומנות וזמינות כמעט תמידית הצלחתי להשיג את הדרך הנכונה והפשוטה לשימוש באלגוריתם.

מאותו רגע, נהניתי להריץ נוסחאות רבות, פשוטות ומורכבות ולראות איך כבמטה קסם מתיר האלגוריתם את כל הסבכים, ויוצר מודל אוטומט לנוסחא.

יתרונו הגדול של האלגוריתם הוא בפשטותו ובהירותו. מפתחי האלגוריתם בעצמם מציעים במאמר מספר דרכים לשיפור וייעול לאלגוריתם אשר הם נמנעו מלהכניס בכדי להשאירו פשוט להבנה ולהוכחה. ככל שהתמקצעתי יותר והכרתי כל שורת קוד לעומק, הרגשתי שאני יכול לחפש ולנסות להוסיף אלמנטים חדשים לאלגוריתם, שירחיבו את יכולת הפעולה שלו (ייתכן מאוד שאם הייתי מקבל את האלגוריתם מורחב ומורכב יותר, היה לי הרבה יותר קשה ללמוד ולהבין אותו). כשמצאתי אלמנט שניתן להוסיף או לשנות, בהתייעצות עם המנחה ותוך בדיקה מעמיקה של מהות השינוי, הוספתי את הצעותיי לעבודה זו.

כיום, בסיום המשימה, אני מרגיש כי הכרתי תחום שהיה זר לי כמעט לחלוטין שחשיבותו בעולמנו ההולך ומתמחשב עצומה. כמובן שזהו רק קצה הקרחון של תחום ידע עצום, אך זכות הייתה לי להכירו.

תודה למנחה העבודה, לחברים שהרצתי עליהם נוסחאות, לצוות המדרשה והפרויקט על העזרה והתמיכה במהלך השנתיים החולפות.

8. ביבליוגרפיה

אדר, נ. (2009). ***מבוא לאימות תוכנה***, מתוך אתר *http://www.underwar.co.il/5-CS/d326*

Gerth, R. & Peled, D. & Vardi, M.Y. & Wolper, P. (1995). ***Simple on-the-fly automatic verification of linear temporal logic***. In: Piotr Dembinski, Marek Sredniawa (Eds.): Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995, pp. 3-18.

Ben-Ari, M. (2001). ***Mathematical Logic for Computer Science***, (2nd ed.), Springer-Verlag, London.

נספחים

נספח 1 - האלגוריתם

```
1 record graph node = [Name:string, Father:string, Incoming:set of string,
2       New:set of formula, Old:set of formula, Next:set of formula];

3 function expand (Node, Nodes_Set)
4       if New(Node)=φ then
5               if ∃ND ∈ Nodes_Set with Old(ND)=Old(Node) and Next(ND)=Next(Node)
6               then Incoming(ND) = Incoming(ND) ∪ Incoming(Node);
7                       return(Nodes_Set);
8               else return(expand([Name ⇐ Father ⇐ new name(),
9                       Incoming ⇐ {Name(Node)}, New ⇐ Next(Node),
10                      Old ⇐ φ , Next ⇐ φ ],{Node} ∪ Nodes_Set))
11      else
12              let η ∈ New;
13              New(Node) := New(Node)\{ η }
14              case η of
15              η = P_n or ¬P_n or η =T or η =F ⇒
16                      if η =F or Neg(η) ∈ Old(Node) (* Current node contains a contradiction *)
17                              then return(Nodes_Set) (* Discard current node *)
18                      else Old(Node):=Old(Node) ∪{η};
19                              return(expand(Node, Nodes_Set));
20              η = μUψ, or μVψ, or μ ∨ ψ ⇒
21                      Node1:=[Name ⇐ new name(), Father ⇐ Name(Node),
22                              Incoming ⇐ ncoming(Node), New ⇐ New(Node) ∪ ({New1(η)}\Old(Node)),
23                              Old ⇐ Old(Node) ∪ { η }, Next=Next(Node) ∪ {Next1(η)}];
24                      Node2:=[Name ⇐ new name(), Father ⇐ Name(Node), Incoming ⇐ Incoming(Node),
25                              New ⇐ New(Node) ∪ ({New2(η)}\Old(Node)),
26                              Old ⇐ Old(Node) ∪ { η }, Next ⇐ Next(Node)];
27                      return(expand(Node2, expand(Node1, Nodes_Set)));
28              η = μ ∧ ψ ⇒
29                      return(expand([Name ⇐ Name(Node), Father ⇐ Father(Node),
30                              Incoming ⇐ Incoming(Node), New ⇐ New(Node) ∪({μ,ψ}\Old(Node)),
31                              Old ⇐ Old(Node) ∪{η}, Next=Next(Node)], Nodes_Set))
32 end expand;
33 function create graph (φ)
34      return(expand([Name ⇐ Father ⇐ new name(), Incoming ⇐ {init},
35              New ⇐ { φ }, Old ⇐ φ , Next ⇐ φ ], φ ))
36 end create graph;
```