גירסה 1.00 – 18.7.2009

שפת LOTOS

ניר אדר

LOTOS

1. תוכן עניינים

1. תוכן עניינים 2
2. הכרות עם LOTOS ומבוא לתהליכים 3
3. קלט ופלט 5
4. עוד על תהליכים 7
5. הגדרת תהליכים 9
6. מקביליות וסינכרון 11
   - 6.1 מקביליות, צורה 1 11
   - 6.2 סינכרון בין תהליכים 12
   - 6.3 מקביליות, צורה 2 – PARTIAL SYNCHRONIZATION 12
   - 6.4 מקביליות, צורה 3 - FULL SYNCHRONIZATION 13
   - 6.5 DEADLOCKS 14
7. נושאים נוספים 15
   - 7.1 הרחבה – EVENT STRUCTURE 15
   - 7.2 דרכים נוספות להרכבה של תהליכים 16
   - 7.3 ביטויי התנהגות (BEHAVIOR EXPRESSIONS) בשפת LOTOS 17
8. סמנטיקה ו-LOTOS 18
9. מה הלאה? 21
10. מקורות 22

2. הכרות עם LOTOS ומבוא לתהליכים

LOTOS (Language Of Temporal Ordering Specifications) – שפה השייכת למשפחת שפות המכונה **Process Algebra**. **התהליך (process)** הוא האלמנט הבסיסי בשפה.

בעולם של LOTOS אנחנו מסתכלים על מערכת מקבילית כעל תהליך, שיתכן שמורכב ממספר תתי-תהליכים פנימיים. **תת-תהליך** הינו תהליך בפני עצמו, ולכן תיאור מערכת באמצעות LOTOS הוא למעשה תיאור והגדרת היררכיה של תהליכים. הפעולה הבסיסית ביותר היא הרכבת פונקציות (תהליכים), אולם ישנן בשפה גם אלמנטים מורכבים יותר לצורך תיאור של עבודה מקבילית.

תהליך הוא ישות המסוגלת לבצע פעילות פנימית לא ידועה, וכן לתקשר עם תהליכים אחרים המצויים בסביבה. תקשורת בין תהליכים מתוארת על ידי יחידות אטומיות של סינכרון להן אנחנו קוראים **אירועים (events)** או **פעולות (actions)**. האירועים נחשבים אטומיים מכיוון שהם מתרחשים בין-רגע, בלי לקחת זמן.

אנחנו מתייחסים אל תהליכים כאל קופסה שחורה – הפעילות הפנימית שלהם מוסתרת ואנחנו מסתכלים רק על התקשורת שלהם עם תהליכים אחרים. התפיסה של קופסה שחורה חוזרת על עצמה רבות בהתייחסות ב-LOTOS.

תהליך מתקשר עם סביבתו על ידי שערים. **שער (gate)** מספק ממשק בין התהליך לסביבה.

תהליך P, כקופסה שחורה עם 4 שערים: a, b, c, d

הסמנטיקה של תהליך מתוארת על ידי **ביטוי התנהגות (behavior expression)**, שיכונה לעתים פשוט **התנהגות**. זהו ביטוי המייצג סדרה חוקית של פעולות נראות. שפת LOTOS מגדירה תהליך על ידי סדרה של אירועים ופעולות, ולא על ידי מצב. בהמשך המסמך אנו נקרא לביטוי התנהגות B גם בשם "תהליך", למרות שאין לו שם, כמו השמות שאנו ניתן בהמשך לתהליכים שנגדיר.

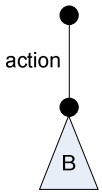
התנהגות טיפוסית מבצעת פעולה, ולאחריה אנו עוברים לתהליך נוסף. התנהגות נראית כמו פונקציה רקורסיבית שלעולם אינה חוזרת אל הפונקציה המקורית.

אופרטור "פעולה מקדימה" – נקודה פסיק:

- אופרטור זה מאפשר לנו להגדיר סדרתיות.
- האופרטור מאפשר לנו להגיד כי התנהגות B מתרחשת בעקבות אירוע action.
- צורת הכתיבה:

(action ; B)

- חשוב לשים לב שבעוד רוב האופרטורים בשפת LOTOS מקבלים 2 התנהגויות כפרמטרים, אופרטור זה מקבל אירוע+התנהגות. כמו כן, התוצאה של אופרטור זה היא ביטוי התנהגות.
- בצורה גרפית:
  - אירועים מסומנים על ידי קו שלידו תווית עם שם האירוע.
  - ביטוי התנהגות מצוין על ידי משולש (ביצוג סדרתי) או קופסה (ביצוג מקבילי).
- נסמן את אופרטור "פעולה מקדימה" כך:

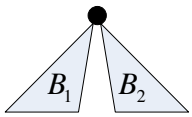


אופרטור הבחירה "[]":

- מפרטים רבים הכתובים ב-LOTOS מערבים **אי-דטרמיניזם**. כדי לציין שניתן לבחור בהתנהגות A או בהתנהגות B על ידי בחירה אי דטרמיניסטית, נכתוב:

(A [] B)

- סימון האופרטור בצורה גרפית:



- בעוד שאופרטור ; עושה את העץ עמוק יותר, אופרטור [] יוצר פיצולים ברמה הנוכחית.

אופרטור [] ואופרטור ; הם האופרטורים הבסיסיים של LOTOS. אופרטורים אחרים בשפה יכולים להיות מבוטאים על ידי שני אופרטורים אלה.

3. קלט ופלט

מופע של קלט/פלט בסביבת LOTOS ממודל על ידי *אירוע* או *פעולה*. הקלט והפלט מגיעים לשערים של התהליכים. לכל שער יש שם המבדיל אותו מהשערים האחרים.

קלט - קבלת ערך מהסביבה:

הפעולה מיוצגת על ידי ביטוי מהצורה:

gate-name ? variable : sort

- פעולת הקלט מיוצגת על ידי קבלת ערך מהסביבה בשער gate-name.
- התהליך יקבל ערך מהסביבה דרך השער gate-name וישמור אותו במשתנה בשם variable.
- sort הוא סוג המשתנה המתקבל. רכיב זה הוא אופציונלי.
- דוגמא לקלט:

  in1 ? age : nat

  nat מציין שמדובר במספר טבעי, ודרך השער in1 נקלוט ערך למשתנה בשם age.
  סוגים נוספים: char, bool

פלט - הצעת ערך בשער:

הפעולה מיוצגת על ידי ביטוי מהצורה:

gate-name ! expression

- expression הוא ביטוי שיהווה את הפלט של השער.
- ה-expression מורכב משילוב של משתנים ושל אופרטורים.
- דוגמא לפלט:

out1 ! (age * age)

כל פעולה המערבת שער שנראה לסביבה (בין אם זו פעולת ! או פעולת ?), ניתנת להפעלה רק אם הסביבה מסוגלת להריץ פעולה המערבת את אותו השער. אם ישנה בחירה אי דטרמיניסטית של פעולות אחרות שכן יכולות להתבצע, אחת מתוכן תבחר. אחרת התהליך אינו יכול להתקדם עד שהסביבה תסכים להריץ פעולות המערבות שערים אלה.

דוגמא לתהליך המציג את מה שלמדנו עד כה:

(ask ? x ; answer ! (x + 1)) [] answer ! 0

התהליך יכול לקבל ערך עבור המשתנה x מהסביבה בשער ask ואז להציע לסביבה את הערך x+1 בשער answer. לחילופין התהליך יכול להציע לסביבה באופן מיידי את הערך 0.

4. עוד על תהליכים

שומר (guard):

לפני פעולה יכול להופיע שומר. שומר הוא פרדיקט המוקף בסוגריים מרובעים שלאחריהם חץ. הפרדיקט חייב לקבל ערך אמת על מנת שיהיה ניתן לבחור לבצע את הפעולה. בעזרת שומרים אנחנו מסוגלים להגביל את הבחירה בפעולות כאשר הפרדיקט לא מתקיים. עם זאת, בחירה עדיין יכולה להיות לא דטרמיניסטית כאשר קיימים כמה שומרים שמסתפקים. לדוגמא:

([nonempty(queue) = true] --> give ! head(queue)) []

([notfull(queue) = true] --> put ? temp)

אנו מניחים בדוגמא זו כי head, nonempty, notfull באים מההגדרה של queue.

מבנה פנימי מוסתר:

כדי לתאר שלתהליך יש מבנה פנימי ופעולות פנימיות שלא רלוונטיים לפעולות הנראות ולשערים המוצהרים, אנו משתמשים באות i. כאשר i מופיעה בפעולה המשמעות היא שמתבצעת פעולה פנימית, ללא ציון פעולה זו. אנו משתמשים באפשרות זו של השפה כדי להגביל את הרזולוציה בה אנו מסתכלים על המפרט.

שערים יכולים להיות מוסתרים באופן מפורש על ידי ההצהרה הבאה:

hide <gate-list> in <behavior>

הפעולות המערבות את השערים שברשימה gate-list יוחלפו על ידי i, ולא יסונכרנו עם אירועים חיצוניים המערבים שערים בעלי שם זה. לדוגמא נביט בקוד הבא:

hide ask in ((ask ? x; answer ! (x + 1)) [] answer ! 0)

ההתנהגות תהיה זהה ל:

(i ; answer ! (x + 1))) [] answer ! 0)

למה נשתמש בזה? הסתרת מבנה פנימי – המערכת החיצונית צריכה לדעת על הקלט והפלט, ולא על שערים פנימיים של תתי-תהליכים.

stop:

התנהגות חשובה נוספת שהינה חלק מ-LOTOS הינה stop. התנהגות זו מייצגת קופסה שחורה מוחלטת: לא ניתן לצפות בשום התנהגות אם התנהגות stop מבוצעת. התנהגות stop יכולה לציין כי המערכת הגיעה ל-deadlock, או שפעולת המערכת הסתיימה.
אם יש stop כחלק מ-choice statement, אופציה זו מתנהגת כמעט כאילו יש לה שומר עם פרדיקט false: אופציה זו נבחרת רק אם אין אף אלטרנטיבה אחרת.

נשים לב שניתן להתייחס ל-stop כאל תהליך מוגדר מראש בשפת LOTOS (ביטוי התנהגות) שאינו מסוגל לבצע אף פקודה או לתקשר עם תהליכים אחרים בשפה.

דוגמא לשימוש:

a; b; c; **stop**

exit:

הפעלת פעולת exit מסיימת את המופע הנוכחי של התהליך בו היא נמצאת. בניגוד ל-stop, אחרי פעולת exit יתכן תהליך נוסף בו הסביבה תמשיך. exit מייצגת סיום תקין של תהליך.

דוגמא לשימוש:

keyboard ? val : nat ; exit(val)

התהליך שמסתיים מחזיר לתהליך הבא את הערך val. הערך val עצמו חייב להיות מאחד מהסוגים שהוגדרו ב-functionality של התהליך המסתיים.

5. הגדרת תהליכים

אנו מסוגלים להגדיר "תהליכים מופשטים" עם פרמטרים פורמלים. לאחר מכן נוכל ליצור מספר מופעים של התהליכים כרצוננו. לכל מופע של התהליך יהיו השערים שלו והפרמטרים שלו.

מופע של תהליך הינו הרצת מופע של **הגדרת תהליך**.

מבחינת תחביר, תהליך מוגדר כך:

```
PROCESS

        proc-name [ gate-names ] (formal-parameters) : functionality :=

                behavior

WHERE

        local-definitions

ENDPROC
```

חלקי ההגדרה:

- ה-header מורכב משם התהליך, שמות השערים והפרמטרים הפורמלים.
- ההתנהגות של התהלים משתמשת בביטויי LOTOS ובמונחים של הפרמטרים הפורמלים והשערים.
- ההגדרות הלוקליות הן אופציונאליות, ואפשר להשתמש בה כדי להגדיר הגדרות שישמשו את הפונקציונליות, או לתיאור תהליכים פנימיים.

פירוט על הרכיבים:

- **functionality** מגדירה את סוגי הערכים שיכולים להיות מועברים לתהליך נוסף לאחר שהתהליך הנוכחי מסתיים בצורה תקינה.
  - אם התהליך לא מתכוון להסתיים (כלומר אין לו התנהגות exit), ה-functionality שלו תהיה מילת המפתח **noexit**.

- אחרת, הפונקציונליות של התהליך הינה מילת המפתח exit ולאחריה רצף של סוגי מידע. בהגדרת התהליך – בכל מקום שבו מתרחשת פעולת exit יש להעביר ערכים המתאימים להגדרה בפונקציונליות.

- **local-definitions**
  - ניתן להגדיר תהליכי משנה ב-local-definitions (תת תהליכים של התהליך הנוכחי)
  - ניתן להגדיר נתונים והגדרות נוספים לשימושינו.
    - הגדרות הנתונים של המערכת נעשות על ידי שימוש בשפה האלגברית ACT-ONE שהינה חלק מ-LOTOS.
    - אלגברת תהליכים מוגדרת על ידי LOTOS ולא על ידי שימוש ב-ACT-ONE.
  - במסמך זה אנחנו לא נציג את ACT-ONE ונשתמש ב-local-definitions לצורך הגדרת תהליכי משנה בלבד.

דוגמא: נדגים תהליך הממדל תעבורה לא אמינה על קו, שמדי פעם נאבד בה ביט. פרוטוקול ברמה גבוהה מבקש שליחה חוזרת של הביט, ואנו נוכל להגדיר:

```
process BADLINE [ Get, Put ] : noexit :=

        Get ? bit : bool  ;

        ( i         ; Put ! bit          ; BADLINE [Get, Put] )

        []

        ( i         ; BADLINE [Get, Put] )

Endproc
```

תהליך לייצוג קו לא אמין. הדוגמא לקוחה מספרו של שמואל כץ

לתהליך 2 שערים. התהליך הינו תהליך שלא מסתיים (noexit). הוא מקבל קלט בקו Get ואז, בהתאם להחלטה פנימי, הוא יכול להעביר את הביט לקו Put, או לא לשלוח דבר ולחזור על התהליך.

ניתן ליצור כמה מופעים של BADLINE עם שערים שונים שמשמעותם ערוצים שונים עם שגיאות. למשל

BADLINE [in_chan, out_chan]

יכול לקשר רכיב שיש לו שער בשם in_chan לשער out_chan כאשר למופעים אחרים של BADLINE לא יהיה כל קשר למופע זה, למעט דרך פעולתם.

נקודה נוספת בה ניתן להבחין היא השימוש ב**רקורסיה**. בשפת LOTOS אין הגבלה על עומק הרקורסיה, ולמעשה שימוש ברקורסיה זו הדרך להגדיר תהליכים שאינם מסתיימים (לולאות אינסופיות).

6. מקביליות וסינכרון

הסביבה של התהליך היא לרוב אוסף של תהליכים אחרים שיכולים לרוץ במקביל. בשפת LOTOS לא קיימת מקביליות אמיתית – אירועים אינם יכולים להתרחש ממש באותו הרגע (כאמור – אירועים הם אטומיים), אך עם זאת מגבלה זו איננה פוגעת בכוח הביטוי של השפה.

LOTOS מציגה 3 סוגים שונים של מקביליות בין תהליכים אותם נציג מיד. ההבדלים ביניהם הינם ברמת הקשר בין התהליכים הפועלים במקביל. (נרחיב על כך מיד).

6.1 מקביליות, צורה 1

הצורה הפשוטה ביותר של מקביליות אינה מערבת סינכרון או תקשורת בין תהליכים, והיא מסומנת על ידי:

A ||| B

הפעולות של תהליך A ושל תהליך B מופרדות: בכל צעד יתכן שתבוצע פעולה של תהליך A, וכן יתכן שתבוצע פעולה של תהליך B. אפילו אם לשני התהליכים שערים זהים הפעולות מבוצעות ללא קשר לפעולות בתהליך השני. במקרה כזה השערים הנראים במערכת הם איחוד של השערים של שני התהליכים.

דוגמא:

```
process bidirect[In1, Out1, In2, Out2] : noexit :=

        BADLINE[In1, Out1] ||| BADLINE[In2, Out2]

endproc
```

מקביליות, צורה 1 – לקוח מסיפרו של שמואל כץ

6.2 סינכרון בין תהליכים

תהליכים יכולים לתקשר ביניהם על ידי סינכרון בשערים נתונים. **סינכרון** פירושו שיתוף שער בין שני התהליכים.

העברת ערך:

- הסוג הנפוץ של אירוע סינכרוניזציה הוא הצעת ערך על ידי תהליך אחד בשער מסויים, וקבלת הערך על ידי התהליך השני. במקרה כזה הצעת המידע מקבילה לפלט, וקבלת המידע מקבילה לקלט. אם לתהליך A יש אפשרות לפעולה G ! 7 ולתהליך B יש אפשרות לפעולה G ? x, וכן G הוא שער ברשימת השערים, אזי 2 האירועים יכולים לקרות בו זמנית ולהחשב "אירוע סינכרון".
- הסביבה מבחינה רק בהתרחשות של אירוע אחד.
- האירוע מתרחש ב-2 התהליכים.
- <u>חשוב:</u> אם אחד התהליכים הגיע למקום ההתקשרות והשני לא – התהליך הראשון יצטרך לחכות לתהליך השני. נקודה זו קריטית כאשר נדבר על deadlock.

קבלת ערך משותף:

- 2 תהליכים יכולים לבקש ערך לאותו שער, לדוגמא G ? x וגם G ? y – במקרה כזה התהליכים יסכימו על ערך מהתחום ויציבו אותו לתוך המשתנים x, y.

6.3 מקביליות, צורה 2 – partial synchronization

מקביליות זו מסומנת על ידי:

A | [<gatelist>] | B

המשמעות של האופרטור:

- השערים הנמצאים ברשימת השערים חייבים להיות מסונכרנים בין A ל-B על ידי **אירוע סינכרוניזציה**.
- כל אירוע הקורה בשער g1 משותף קורה באותו הרגע ב-2 התהליכים. A ו-B אינם יכולים להתנהג יותר כישויות נפרדות במובן זה.
- שערים שאינם ברשימת השערים מופרדים – פעולות בהן לא משפיעות אחד על השני.
- מקביליות מצורה 1 היא למעשה מקביליות מצורה 2 בה רשימת השערים ריקה.

- דוגמא:

```
( keyboard ? num : nat ; line ! num ; ... )
| [line ] |
( line ? var : nat ; screen ! var ; ... )
```

- דוגמא נוספת:

```
procA[a, b]

| [a] |

procB[a, c]
```

במקרה זה האירועים בשער a קורים תמיד ב-2 התהליכים בגלל ש-a ברשימת השערים המסונכרנים. אירועים בשערים b, c יכולים לקרות באופן עצמאי ב-ProcA או ב-ProcB.

6.4. מקביליות, צורה 3 - Full Synchronization

- אופרטור סינכרון נוסף הוא **אופרטור סינכרון מלא** המסומן על ידי ||
- סינכרון מלא פירושו שיתוף של כל השערים של התהליכים, בין אם הם שערים משותפים ובין אם לא.
- סינכרון מלא פירושו גם סינכרון של כל האירועים הגלויים.
- סינכרון מלא הוא סינכרון חלקי (צורה 2) כאשר כל השערים משותפים.

סנכרון מלא מייצג הסכמה מלאה של שתי ההתנהגויות כדי לקבל אירוע מהסביבה. אם האירועים המוצעים על ידי 2 ההתנהגויות בשערים שונים – הם לא יתרחשו.

אם אירוע מוצע על ידי שתי ההתנהגויות באותו הזמן, האירוע עשוי להתרחש ולשנות את 2 ההתנהגויות. אם B_1 מציע אירוע a ומשנה את מצבו ל- B_1' , ו- B_2 מציע אירוע a ומשנה את מצבו ל- B_2' , אזי איחוד התהליכים B_1 ו- B_2 על ידי מקביליות בצורה 3 עשוי לקבל אירוע a ואז להתנהג כ- B_1' המצוי בסינכרון מלא עם B_2' .

6.5. Deadlocks

נאמר שתהליך (התנהגות) הינו deadlock כאשר לא ניתן לצפות בשום פעולה מצידו. Deadlock מתרחש כאשר התנאים הבאים מתקיימים:

- אין אירועים נפרדים המתרחשים.
- אירועי הסינכרוניזציה המופיעים לא מתאימים.

במערכות אמיתיות לרוב נמנע מ-deadlocks ככל שניתן.

ב-LOTOS הפקודה stop משמשת אותנו לתיאור סיום פעולת המערכת או הגעה ל-deadlock.

7. נושאים נוספים

7.1 הרחבה – Event Structure

עד כה ראינו אירועים המקבלים רק ערך אחד מהפעולה. למשל:

Get ? x

המשמעות שהשער Get קולט ערך ושם אותו לתוך המשתנה x.

נציג כרגע הרחבה המאפשרת לקבל/לפלוט מספר כלשהו של ערכים במהלך האירוע. אנחנו בעצם קולטים tuple של ערכים.

Event Structure – רשימה סדורה של הערכים המתקבלים באירוע.
בדרך כלל כל האירועים שיכולים לקרות בשער ספציפי הם מאותו ה-event structure.

דוגמאות:

Event	Event Structure
Bus ! 0 ! 1 ! 1 ! 0 ! 0	Bit, Bit, Bit, Bit, Bit
Terminal ? ID : nat ? Amount : nat	nat, nat
Port ! id ? Message : string	nat, string

מה ההבדלים בין קבלת מספר קלטים לבין קבלת tuple של ערכים? ההבדל הוא באטומיות. במקרה של Tupple מתרחש רק אירוע אחד בו נקלטים כל הערכים, ולא מספר אירועים.

7.2. דרכים נוספות להרכבה של תהליכים

תהליך מורכב מביטויי התנהגות, המורכבים יחדיו על ידי נקודה פסיק ועל ידי אופרטורי בחירה. לאחר שתהליך מוגדר אנחנו גם יכולים ליצור תהליכים מורכבים יותר על ידי רקורסיית זנב ומקביליות.

נציג כעת 2 דרכים נוספות להרכבת תהליכים ב-LOTOS:

sequential composition (enabling)

כמו שכבר ציינו בהגדרה של exit, שני תהליכים יכולים לפעול באופן סדרתי, אחד אחרי השני. הכתיבה הבאה אומרת כי אחרי ש-P מסתיים באופן תקין (על ידי exit), אז ההמשך הוא התנהגות R:

P >> R

ההתנהגות שמחברת בין הסיום של P להתחלה של R מסומנת על ידי הפעולה הפנימית i.

כדי לציין ש-R מקבלת ערכים (לתוך משתנים) מהסיום של P, הכתיבה המלאה היא:

P >> accept variable-list in R

כאשר variable-list זו רשימה של משתנים והסוגים שלהם שבהם נשתמש בתוך R, והם חייבים להתאים בדיוק ל-functionality של P. אפשר להסתכל על הרכבה זו כעל מקרה מיוחד של סינכרון, בה הסנכרון מתרחש כאשר התהליך הראשון מגיע להתנהגות exit. הסינכרון נסתר מהסביבה מכיוון שהמעבר מוגדר כפעולה פנימית.

Disabling

הסוג השני של הרכבת תהליכים מייצג הפסקת התנהגות אחת על יד התנהגות אחרת. המשמעות היא שהתנהגות P עשויה בכל רגע להמשיך כהתנהגות R. נסמן זאת כך:

P [> R

בצורה גרפית העץ שמייצג את R מוסף בכל נקודה בעץ P כבחירה אי-דטרמיניסטית, למעט בעלים בהם יש הצהרת exit.

כאשר P מסתיים בצורה מוצלחת אין אפשרות להמשיך כ-R. בנוסף, במידה ועברנו להתנהגות R – ההתנהגות לא תחזור להיות P.

7.3. ביטויי התנהגות (behavior expressions) בשפת LOTOS

הטבלה הבאה מציגה את התחביר של כל ביטויי ההתנהגות החוקיים בשפת LOTOS.

B, B1, B2 הם ביטויי התנהגות כלשהם.

NAME	SYNTAX
inaction	stop
action prefix - unobservable (internal) - observable	 i ; B g; B
choice	B1 [] B2
parallel composition - general case - pure interleaving - full synchronization	 B1 |[g1, ... , gn]| B2 B1 ||| B2 B1 || B2
hiding	hide g1, ... , gn in B
process instantiation	p [g1, ... ,gn]
successful termination	Exit
sequential composition (enabling)	B1 >> B2
Disabling	B1 [> B2

8. סמנטיקה ו-LOTOS

המטרה שלנו בהתעסקות בסמנטיקה היא על מנת שנוכל להגיד "אילו תהליכים הם זהים". התשובה לשאלה זו תוכל לעזור בהבהרת בעיות של מקביליות.

לצורך הצגת הסמנטיקה אנחנו עובדים עם גרסה מצומצמת של LOTOS הנקראת BASIC LOTOS. בגרסה זו אין סימני ? ו-! – אנחנו מתרכזים בסנכרון בין התהליכים בלבד.

Trace Semantics:

- תהליך מוגדר על ידי אוסף כל סדרות הצעדים הנראות (פעולות) האפשריות שהוא עושה.
- שני תהליכים זהים אם אוסף הסדרות הנראות שלהם זהה.
- לפי סמנטיקה זו, למשל, 2 התהליכים הבאים זהים (אוסף כל סדרת הצעדים הנראות שלהם הוא זהה):

  G1 = (a; b; c; G1) [] (a; b; d; G1)

  G2 = a; ((b; c; G2) [] (b; d; G2))

- הגדרה זו של סמנטיקה לא מספיק טובה לנו בעבודה ב-LOTOS. כאשר תהליך הוא אבן בניה לתהליכים אחרים ומסתנכרן עם תהליכים אחרים, ניתן למצוא תהליכים זהים לפי Trace Semantics, שאחד מהם יצליח להסתנכרן עם תהליכים אחרים והשני יכשל. לפיכך, נציג סמנטיקות נוספות.

Testing/Refusal/Failure Semantics

- 2 תהליכים A, B הם נבדלים אם ניתן להגדיר תהליך P ותרחיש בו P יסתנכרן עם A בהצלחה, אך לא עם B.
- 2 תהליכים A, B הם זהים אם לא ניתן למצוא תהליך P כזה.
- למעשה מודל של קופסה שחורה – אם 2 תהליכים מגיבים לתהליך אחר בדיוק באותה הצורה – הם זהים.

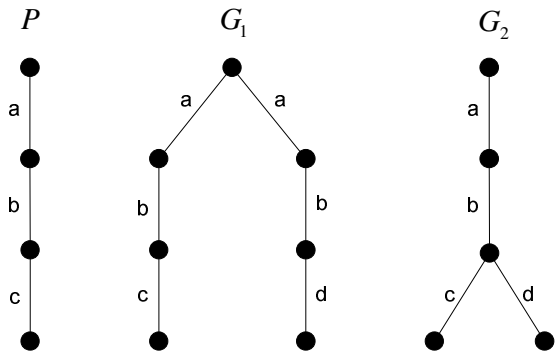
- דוגמא:

G1 = (a; b; c; G1) [] (a; b; d; G1)

G2 = a; b; ((c; G2) [] (d; G2))

P = a; b; c; P

נצייר את התהליכים בצורה גרפית:



מתקיים: G2 || P אבל לא מתקיים: G1 || P

במקרה של P ו-G2, שני התהליכים יסכימו על a, לאחר מכן יסכימו על b, ואז יש פעולה ששניהם מסוגלים להסכים עליה, שהיא c. במקרה של G1 ו-P, שני התהליכים יסכימו על a, אבל במידה וב-G1 נבחר את המסלול הימני נגיע למצב של deadlock בו לא תהיה פעולה עליה יסכימו שני התהליכים.

Bisimulation Semantics

- לפי סמנטיקה זו – שני תהליכים זהים אם העצים המתארים אותם זהים. (עד כדי הפעולות המתרחשות בתוך i).

השוואה בין הסמנטיקות השונות:

- Trace היא הסמנטיקה החלשה ביותר, אחריה Testing Semantics ואילו Bisimulation היא הסמנטיקה החזקה ביותר. הכוונה ב"חלשה" ו-"חזקה" הוא שככל שהסמנטיקה חזקה יותר, היא מבדילה יותר בין תהליכים – יותר תהליכים לא יחשבו בה זהים.
- Trace שימושית לבדיקות על המערכת כולה. Trace מסתכלת על אוסף כל החישובים של התוכנית – כלומר – איך התוכנית יכולה להתנהג.
- Testing פופולארית כאשר מתעסקים ברכיבים ומבצעים בדיקות קופסה שחורה. סמנטיקה זו מאפשרת לנו להחליף רכיב במערכת ברכיב שיתנהג בצורה זהה.

9. מה הלאה?

זהו סוף המסמך על שפת LOTOS. מסמך זה הציג בקצרה את שפת LOTOS וחלק ממרכיביה. הדגש שלי במסמך היה על הבהירות והעברת הרעיונות, ופחות על השימוש הפרקטי בשפה. על מנת לכתוב בה ולהבינה יותר, אמליץ במיוחד על המקורות הבאים:

1. **"An Introduction to LOTOS" מאת Arturo Azcorra Salona, Juan Quemada Vives, Santiago Pavon Gomez.**
   מקור זה היווה את הבסיס העיקרי למסמך. מלבד התוכן שבמסמך זה, הוא מכיל נושאים נוספים בשפה שלא הוזכרו במסמך זה.
   כמו כן, הוא מכיל דוגמאות רבות נוספות לכל הנושאים שהוצגו במסמך – מומלץ לעבור עליו ועל הדוגמאות השונות.

2. **"Formal Specification Book", Prof. Shmuel Katz**
   הספר של שמואל כץ מציג את החומר בזוית קצת שונה מזו של 3 הכותבים של המקור הראשון.
   אחרי קריאת המקור הראשון, אמליץ לעבור על הפרק הרלוונטי בספר להעמקה נוספת.

ניר אדר

יולי 2009

10.מקורות

1. http://en.wikipedia.org/wiki/Process_calculus
2. http://www.spatial.maine.edu/~worboys/processes/baeten%20history%20PA.pdf
   "A Brief History of Process Algebra", J.C.M. Baeten, Department of Computer Science, Technische Universiteit Eindhoven, The Netherlands
3. http://lotos.csi.uottawa.ca/ftp/pub/Lotos/Intro/BB-LotosTutorial.pdf
   "Introduction to the ISO Specification Language LOTOS", 1989, Tommaso Bolognesi, Ed Brinksma
4. http://www.cas.mcmaster.ca/~sartipi/course/cas707/w07/slides/Mar06-LOTOS.pdf
   "LOTOS: Language of Temporal Ordering Specification", 2007, Dr. Kamran Sartipi
5. "An Introduction to LOTOS", 1993, Arturo Azcorra Salona, Juan Quemada Vives, Santiago Pavon Gomez
6. “Formal Specification Book”, Prof. Shmuel Katz