Assignment 3

:1
כן הצורה let היא צורה מיוחדת בשפה L3. בהגדרת חוקי השפות -צורה היא מיוחדת אם יש לה חישוב של תחביר וסמנטיקה מיוחדים .נוכל לראות בL3 גם ב parser וגם ב interpeter כי למילה let יש לאורך כל הזרימה של התכנית צורה מיוחדת משלה .נוכיח : ב L3-ast קיימת הפונקציה makeLetExp והיא צורה מיוחדת לlet .

:2
המרת value לביטוי מסוג CExp . הסיבה היא שבשלב שהפונקציה value2litexp מופעלת כל הביטויים בתוך גוף הפונקציה חושבו כבר מCExp ל value. בשיטת החישוב substitution אנו נאלצים לחשב שוב את כל הערכים בגוף הפונקציה ע"מ לשלבם בצורה נכונה וזאת ע"י שילובם מחדש בעץ הAST . בעץ ה AST כל הערכים הם CExp ולא value ולכן תהיה פה בעיה של טיפוסים אם לא נשנה אותם בצורה ההפוכה ממה שעשינו. בפועל פעולה זו לוקחת value ועוטפת אותו באוביקט cExp מאותו סוג .

:3
בצורת החישוב normal גוף הפונקציה מחושב להיות value רק במידה ויש בו שימוש מיידי, אם לא חישובו נדחה לשלב מאוחר יותר . בזמן הפעולה L3normalApplyProc גוף הפונקציה עדיין ערך מסוג CExp ולא Value ולכן תאימות הטיפוסים תקינה בהצבה בAST ופעולה זו לא נדרשת.

:4
בפונקציית valueToLitExp ביצענו המרה של ערכים למשתני CExp כי נדרשנו להמירם ע"מ לבצע הצבה בגוף הפונקציה בזמן ההפעלה . במודל הסביבות בגלל השימוש במבנים כהיררכיה של פריימים (ולא כמו שערכי המשתנים שמורים בסביבה הגלובלית בלבד) אין לנו צורך בהצבת הערכים בתוך גוף הפונקציה ע"י חיפושם בסביבה הגלובלית כערכי varRef מסוג CExp יותר מזה – אין הצבה כלל ,ואין צורך גם לשנות את שמות המשתנים לייחודים – נבצע זאת ע"י מציאת המופע הראשון של הגדרת המשתנה בהיררכת הפריימים ומשם ניקח את הערך למשתנה והוא הרלוונטי היחיד בשביל גוף הפונקציה הנ"ל.

:5
בחישוב מסוג applicative נחשב את כל הביטויים גם אם אין בהם צורך ולא יהיה בהם כל שימוש.
חישובים אלו עולים זמן ריצה מיותר ולכן בחישוב מסוג normal נחשב את הביטוי להיות ערך רק אם יש דרישה לשימוש בערך הזה.

```
(
 (if (< 2 1)
   (+ 5 1)
   (- 2 1)
)
```

בקוד כזה האינטרפרטר בשיטת הנורמל לא יחשב את הביטוי (1 5 +) כי בהפעלת ה-if בכל סביבה ומצב יחושב רק אחד מהביטויים בלבד. לעומת זאת בשיטת apllicative יחושבו כל הערכים כולל (1 5 +) שיחושב באופן מיותר.

:6
במידה ויש כמה חישובים מאותו סוג שנדרש לבצע בשיטת ה apllicative נבצע אותם פעם אחת ולאחר החישוב הם יקבלו את הערך של התוצאה . לעומת זאת בשיטת ה normal נוכל לדחות את החישוב לשלב מאוחר יותר ובשל כך יידרש לבצע אותו מספר רב של פעמים מאשר מה שהיינו יכולים אם היינו מחשבים בשלב מוקדם יותר.

```
)
(lambda (x) (+ x x))
(* 2 3)
)
```

אפליקטיב: נחשב תחילה את (* 2 3) ונציב

```
(+ 6 6)
```

נורמל: נציב (* 2 3) מבלי לחשבם בשלב זה. הם יחושבו מאוחר יותר כאשר זה יידרש עבור הפעלת האופרטור הפרימיטיבי +.

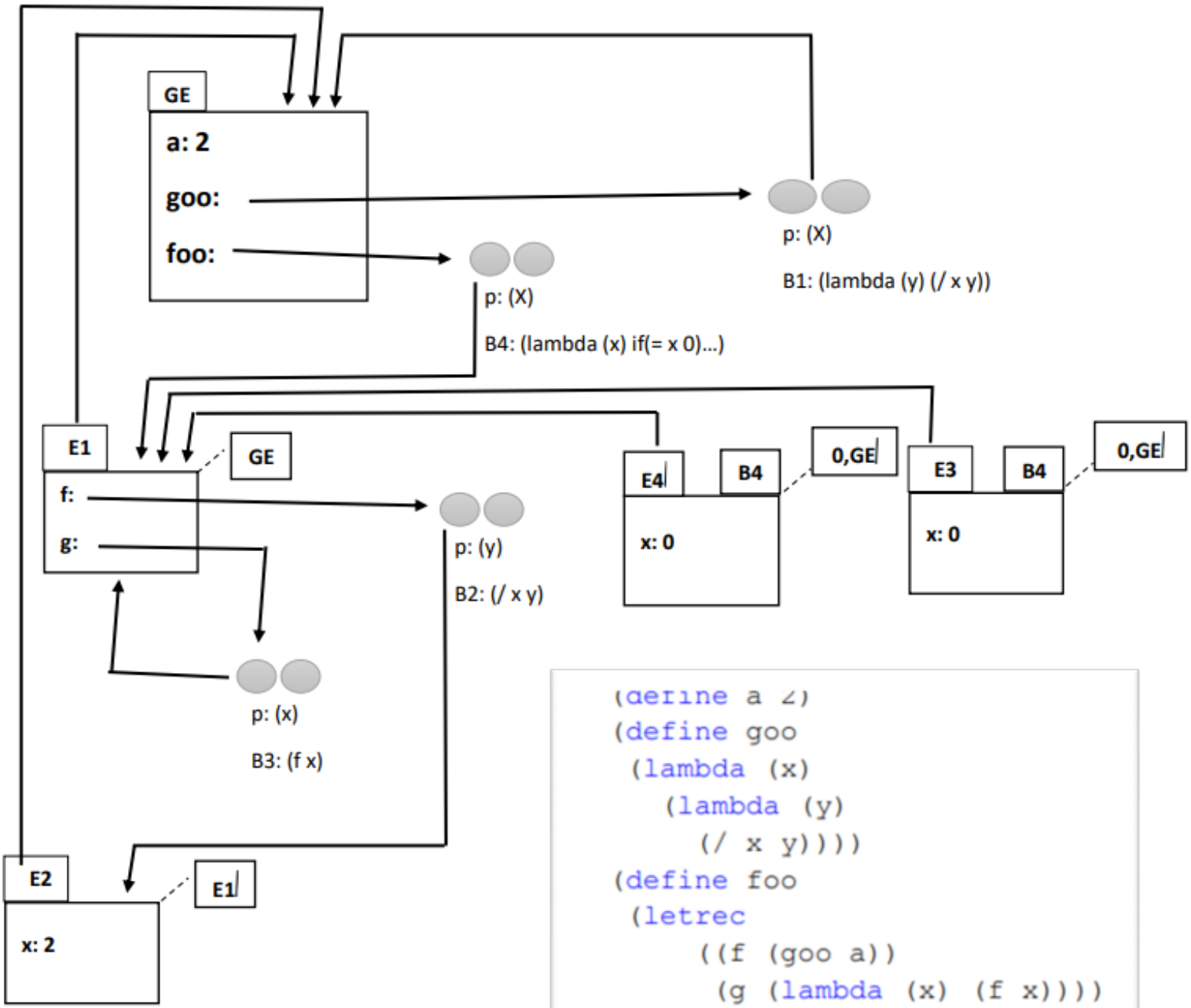
```
(+ (* 2 3) (* 2 3))
```

בדוגמא זו, אפליקטיב יעיל יותר, בנורמל החישוב של (* 2 3) יבוצע פעמיים.

:7
א:
ב"מודל ההצבה", הצבה מתבצעת רק על משתנים חופשיים. במצב בו כל המתשנים קשורים אז בכל פעם שנבצע פעולת הצבה בתוך גוף הפונקציה של הפרמטרים, נבדוק האם המשתנה חופשי – נגלה שלא כי במקרה שלנו אין משתנים חופשיים ולכן נמשיך ללא ביצוע renaming. לכן ביצוע ה-renaming מיותר. במצב כזה לא יהיה בלבול בין שמות המשתנים כי המשתנה קשור והוא יוצב רק מול ה-vardecl שלו.
דוגמה – בפונקציה (lambda(x) (lambda(x) (+ 5 x) 2) 3) אין משתנים חופשיים. לכן בהצבה הראשונה של x = 3 אין משתנה x חופשי בגוף הפונקציה (lambda(x) (+ 5 x)). מכאן לא יוצב הערך 3 באף אחד מהמשתנים והערך x = 2 יוצב לאחר מכן בביטוי (+ 5 x). לכן אין חשש להצבה לא נכונה במקרים כאלה.

ב:
בכל הפעולות שהם לא אבלואציה של ערך מסוג applyProcedure נעבוד כפי שהוצג בכיתה.
את פעולת ההחלפה הנאיבית בביטוי מסוג applyProcedure נבצע באופן הבא:

- נקבל closure, מערך של ערכים (שהם ארגומנטים של הפונקציה שחושבו) וסביבה.
- ניצור משתנה שנקרא Vars שלתוכו נכניס את פרמטרי ה-closure (closure.param).
- ניצור משתנה שנקרא litArgs אשר יבצע החלפה של המערך ערכים שקיבלנו למערך של litExp.
- ניצור משתנה שנקרא body אשר מקבל את גוף ה-closure.
- נחזיר את האבלואציה של פעולת ההחלפה באמצעות הממשתנים שיצרנו לעיל.




```
(define a 2)
(define goo
 (lambda (x)
   (lambda (y)
     (/ x y))))
(define foo
 (letrec
     ((f (goo a))
      (g (lambda (x) (f x))))
   (lambda (x)
     (if (= x 0)
         x
         (g x)))))
(foo (foo 0))
```