מגישים:

דור כרמי- 205789662

עודד בסרבה- 315564641

עבודה 3 ב-PPL:

תשובות לשאלות מהעבודה:

שאלה 1:

1. באופן עקרוני, let הוא ביטוי מיוחד- special form בשפות אותן למדנו- ביטוי שיש עבורו טיפול מיוחד ב-Parser. ניתן לראות זאת ע"י המימוש המיוחד לזיהוי וטיפול בביטוי let, אשר מופיע בתוך הפונקציה parseL3SpecialForm ואשר קיים עבורו ביטוי מיוחד בעץ AST של השפה. אבל בפונקציה eval אין בדיקה מיוחדת ל-LetExp, כלומר אין פירוש סמנטי לביטויים כאלה. לכן זה לא מוגדר כביטוי מיוחד שניתן להחזיר לו ערך, ולכן זה לא מוגדר כ- special form שאפשר להשתמש בו.

```
const L3applicativeEval = (exp: CExp, env: Env): Result<Value> =>
    isNumExp(exp) ? makeOk(exp.val) :
    isBoolExp(exp) ? makeOk(exp.val) :
    isStrExp(exp) ? makeOk(exp.val) :
    isPrimOp(exp) ? makeOk(exp) :
    isVarRef(exp) ? applyEnv(env, exp.var) :
    isLitExp(exp) ? makeOk(exp.val) :
    isIfExp(exp) ? evalIf(exp, env) :
    isProcExp(exp) ? evalProc(exp, env) :
    isAppExp(exp) ? safe2((rator: Value, rands: Value[]) => L3applyProcedure(rator, rands, env))
        (L3applicativeEval(exp.rator, env), mapResult(rand => L3applicativeEval(rand, env), exp.rands)) :
    isLetExp(exp) ? makeFailure('"let" not supported (yet)') :
    makeFailure(`Bad L3 AST ${exp}`);
```

2. שגיאות סמנטיקה שיכולות לקרות כשמריצים תכנית L3:
   a. ריבוי של שמות משתנים עם אותו שם (תחת אותה סביבה):
      i. (lambda (**x x**) (+ (* 3 x) (* 2 x)))
   b. שם לא חוקי של משתנה:
      i. define **if** 5
   c. הגדרת של פעולה חשבונית לא חוקית
      i. (define f (/ 3 0))
   d. שימוש ב-VarRef שלא הוגדר:
      i. (define x f(5))
      ii. f not defined
3. השימוש בפונקציה valueToLitExp הוא כדי להעביר ביטוי שמוגדר כערך אטומי בשפה (value) ל-CExp כדי שנוכל להעביר אותו לפונקציה substitute, כך שתחליף את כל המופעים של המשתנה לביטוי הרצוי, באופן ששאר חלקי הקוד יוכלו להשתמש בו. כלומר אם מוגדר בסביבה שאנחנו נמצאים:

   x=5

   ועכשיו אנחנו רוצים לפרש משמעות של ביטוי שמשתמש ב-x:

   (+ x 3)

נרצה לבדוק מה הערך שלו. וכדי שנוכל להחליף אותו לקודקוד מוגדר בעץ AST ולהמשיך לעשות בו שימוש ב-evaluation, נרצה להעביר אותו כ-CExp, לכן נעביר את הביטוי להיות מביטוי מסוג ערך (value) לביטוי מסוג CExp שה-AST מכיר.
בכדי למנוע את ההמרה הזאת, כל שיש לעשות הוא להגדיר את SExpValue להיות סוג של CExp, כלומר:

CExp=(כל מה שהוא היה קודם)|SExpValue

ואז ניתן להעביר אותו כדי להעריך את ערכם של משתנים אותם אנחנו מחפשים ולהחליף את הביטוי שלהם ב-substitute בערך value שלהם.
השינויים ב-interpreter:
בפונקציה applyClosure: לא נצטרך להעביר את הביטויים לייצוג Cexp ונוכל לשלוח אותם לפונצקציה substitute כמו שהם, שתחליף את כל המופעים של המשתנים הנדרשים בביטוי החדש- שהוא הערך עצמו.

4. באוולואציה נורמלית החישוב (eval) של הביטוי מתבצע רק כאשר ניגשים להשתמש בו, בניגוד ל-applicative שבו הביטוי מחושב מההתחלה. לכן הביטויים נשמרים בחלקי העץ (AST) כביטויי Cexp לא מפורשים, והפירוש שלהם מתבצע רק כאשר זה נדרש. מכאן שאין צורך להחזיר ביטוי מסוג ערך לביטוי מסוג Cexp, שכן כל הביטויים הם Cexp עד אשר מגיעים לשלב שמפרשים אותם לביטוי ערך אמיתי על מנת לחשב את הערך של הקוד אותו אנחנו מפרשים/מפעילים.
5. דוגמאות:

<u>דוגמא שבה applicative מהיר יותר מ-normal:</u>

```
Define x 10

Define y 3

Define myF (lambda () (x*y))

((> (+ myF myF) 0)? myF:0))
```

ב-normal החישוב של myF יתבצע כמה פעמים- בכל פעם שנקרא לפונקציה נצטרך לחשב את הערך שלה.

ב-applicative נחשב אותה בפעם הראשונה ולאחר מכן נציב את הערך בכל מופע, לכן החישוב יתבצע רק פעם אחת.

מכאן מהירות הביצוע של הביטוי הנ"ל בשיטת ה-applicative יהיה מהיר יותר.

<u>דוגמא שבה normal מהיר יותר מ-applicative:</u>

```
(define firstF (lambda (x y) (x*x+y*y+x+y+x+y)))

(define j 0)

((= j 0) ? (1) : (firstF(50,3)))
```

בביטוי הנ"ל ניתו לראות שבחישוב applicative נצטרך לחשב כל אחד מהביטויים שבתוך פונקצית ה-if ורק לאחר מכן להחליט מה התוצאה של הפעלת ה-if.

בחישוב בצורת normal ניגש לחשב את הביטוי בהתאם לתנאי, ולא נחשב את הביטוי השני. במקרה הזה נצטרך רק להחזיר 1 ולא לחשב את הכפל והחילוק שנדרש בביטוי השני, ולכן החישוב יהיה מהיר יותר.

תשובה לשאלה 3.1:

הבעיה היא שהמימוש של האיוולואציה הנורמלית בL3 מתבצע המשתנים ממופים לערכים ולא ממופים לCexp ולכן כאשר אנו יוצרים binding אנו בעצם צריכים לעשות אוולואציה מלאה לביטוי במקום לקשר את המשתנה לביטוי עצמו.