חלק 1

1. הביטוי let הינו special form משום שיש עבורו חוק חישוב מיוחד אשר אינו מוגדר באינטרפטר. חוק החישוב עבור ביטוי let הינו: חישוב הערכים ב bindings ע"פ הסביבה הנוכחית, הגדרת המשתנים ב binding כך שהם מקושרים לערכים שחושבו ולבסוף חישוב ה body ע"פ הסביבה הנוכחית הכוללת את המשתנים הלוקאליים.

2. א. חלוקה ב 0 : (/ 5 0)

ב. אופרנד לא חוקי בהפעלת פרוצדורה : (+ 2 #t)

ג. מספר לא נכון של אופרנדים : ((lambda (x y) (* x y)) 6 7 8)

ד. אופרטור לא חוקי בהפעלת פרוצדורה : (1 2 3)

3. 3.1 נוסיף ל disjoint union שמגדיר את CExp את ה type: SExpValue.
Type זה יוגדר ע"פ ה disjoint union:

```
SExpValue = number | boolean | string | PrimOp | Closure | SymbolSExp |
 EmptySExp | CompoundSExp;
```

כפי שמוגדר ב L3-value.

3.2 נוסיף באינטרפטר בדיקת טייפ SexpValue שבמידה ויוצאת חיובית תחזיר את הביטוי עצמו משום שהוא כבר ערך מחושב. כלומר בפונקציה:

```
const L3applicativeEval = (exp: CExp, env: Env): Result<Value> =>
```

נוסיף:

```
isSExpValue(exp) ? makeOk(exp) :
```

3.3 לדעתנו עדיף להשתמש בפונקציה valueToLitExp מכיוון שהיא יוצרת הפרדה בין הפארסר לאינטרפטר שהם שני תהליכים נפרדים, וככה שינוי של אחד לא משפיע על האחר.

4. ב normal evaluation strategy interpreter ההחלפה מתבצעת לפני חישוב ערך הארגומנטים, ולכן לא נזדקק לפונקציה valueToLitExp שמטרתה להפוך את ה Value שחוזר מחישוב ערך הארגומנטים ל Exp שהוא הטיפוס שלו מצפים קוקודי ה AST.

5. <u>normal מהיר יותר:</u> (L4 ((lambda (x y z) (if x y z)) #f (* 2 3) 5))

ב normal החישוב של ה then לא יבוצע בניגוד ל applicative.

<u>Applicative מהיר יותר:</u> (L4 ((lambda (x) (+ x x)) (* 4 3)))

ב applicative החישוב של (* 4 3) מבוצע פעם אחת בניגוד ל normal שם מבוצע פעמיים.

חלק 3.1

```
#lang lazy

(define x (-))
x
```

```
Welcome to DrRacket, version 7.7 [3m].
Language: lazy, with debugging; memory limit: 128 MB.
#<promise:x>
>
```

נוצר binding בין x ל -, אולם מכיוון שלא הייתה הפעלה של x, הערך של x הינו promise כפי שמוגדר באסטרטגיית lazy. לא נוצרה שגיאה מכיוון שבאסטרטגיה זו מחושב הערך האמיתי של משתנה רק בעת הצורך, ועד אז הוא שמור בתוך מבנה ה promise.

```
#lang lazy

(define x (-))
1
```

```
Welcome to DrRacket, version 7.7 [3m].
Language: lazy, with debugging; memory limit: 128 MB.
1
>
```

בדומה למקרה הקודם, הערך של x הינו promise ולא נוצרת שגיאה מכיוון שהערך האמיתי מחושב רק בעת הצורך. הביטוי '1' מקבל את הערך האמיתי שלו 1 לפי השפה.

```
{ tag: 'Ok', value: 0 }
{ tag: 'Ok', value: 0 }
{ tag: 'Failure', message: 'Type error: - expects numbers ' }
{ tag: 'Failure', message: 'Type error: - expects numbers ' }
{ tag: 'Failure', message: 'Type error: - expects numbers ' }
```

הבעיה במימוש של define היא שכאשר התוכנית יוצרת binding בין var ל – val, היא מחשבת את הערך של val במקום לשמור אותו כ CExp ולחשב את ערכו רק כשיידרש, כפי שנממש בסעיפים הבאים.