אוניברסיטת
אריאל
בשומרון

פקולטה: מדעי הטבע

מהלקה: מדעי המחשב ומתמטיקה

שם הקורס: שפות תכנות

קוד הקורס: 2-7036010-1/2

תאריך בחינה: 05/06/2014 סמ' ב' מועד א'

משך הבחינה: 3 שעות

שם המרצה: ערן עמרי

חומר עזר: אסור

שימוש במחשבון: לא

הוראות כלליות:

- כתבו את תשובותיכם בכתב קריא ומרווח.
- בכל שאלה או סעיף, ניתן לכתוב – **לא יודע/ת** (מבלי להוסיף דבר מעבר לכך) – ולקבל 20% מהניקוד על השאלה או הסעיף.
- אפשר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם.
- יש לענות על כל השאלות.
- ניתן להשיג עד 107 נקודות במבחן.

אוניברסיטת
אריאל
בשומרון

שאלה 1 — BNF — (28 נקודות):

סעיף א' (7 נקודות):

הסבירו את מושג הרב משמעיות (ambiguity) עבור BNF. תנו דוגמה לדקדוק (כלשהו) אשר אינו חד-משמעי (כלומר, הוא סובל מ-ambiguity). הדגימו מדוע הדקדוק שלכם אינו חד-משמעי.

סעיף ב' (8 נקודות):

רוצים להוסיף לשפת הביטויים האריתמטיים הבסיסית AE את האפשרות להשתמש בזכרון – בעזרת פעולות:

set – הכנסת תוצאת חישוב לזכרון.

get – שליפת הערך מהזכרון (בכל שלב נשמר ערך יחיד בזכרון).

בשאלה זו נטפל רק בכתיבת BNF עבור שפה זו (נקרא לשפה זו 'MAE').

פתרון נאיבי מציע את הדקדוק הבא:

```
<MAE> ::= <num>
        | { + <MAE> <MAE> } ; Rule 1
        | { - <MAE> <MAE> } ; Rule 2
        | { * <MAE> <MAE> } ; Rule 3
        | { / <MAE> <MAE> } ; Rule 4
        | { set <MAE> }     ; Rule 5
        | get               ; Rule 6
```

כאן הכוונה בביטוי {set E} (באשר, E הינו ביטוי כלשהו) הינה לחשב את הערך של E ולהכניס ערך זה לזכרון.

הביטוי הבא מדגים בעיה בדקדוק המוצע מעלה:

```
{* {+ {set 1} {set 2}} get}
```

- הראו כיצד נגזר הביטוי מהדקדוק המוצע.
- הסבירו מה הבעיה שמודגמת בביטוי זה.

סעיף ג' (10 נקודות): מלשון להראות

בכדי לפתור את הבעיה הקודמת ובכדי לתת לביטויים בשפה תאימות גדולה יותר לאופן שבו אנחנו משתמשים בזכרון במחשבון, הגדירו דקדוק MAE המקיים את התנאים הבאים:

- ביטוי בשפה הינו **סדרה**, לא ריקה, של תתי ביטויים המתארים חישוב. הביטוי כולו מתחיל בסימן הפעולה **seq** ועטוף בסוגריים מסולסלים.
- כל אחד מתתי הביטויים הללו (מלבד האחרון בסדרה) מתחיל בסימן הפעולה **set**, יש לו אופרנד יחיד והוא עטוף בסוגריים מסולסלים. האחרון בסדרה נראה כמו אופרנד (כמתואר מטה). הראשון אינו כולל את סימן הפעולה **get**.
- **אופרנד** של תת-ביטוי הינו אחת משלוש אפשרויות:
  - **מספר**
  - ביטוי אריתמטי עטוף בסוגריים מסולסלים עם אחד מארבעת סימני הפעולה +, -, /, * ושני אופרנדים.
  - סימן הפעולה **get**. שימו לב! אסור ל-get להופיע בביטוי הראשון בסדרה כולה.

דוגמאות:

```
;; valid sequences  -                    תקין
{seq {set {+ 8 7}}
     {set {* get get}}
     {/ get 2}}
{seq {- 8 2}}
{seq 25}
;; invalid sequences  -                  לא תקין
{seq {set {* 8 get}}      ; cannot begin with a `get'
     24}
{seq {* 8 7}              ; must be a `set'
     24}
{seq {set {+ 1 2}}
     {set {- get 2}}}     ; cannot end with a `set'
{seq {* 2 {set {+ 1 2}}} ; `set' must be outside
     {- get 2}}
```

סעיף ד' (3 נקודות):

הראו כיצד ניתן לגזור ביטוי בשפה של הדקדוק שהגדרתם. בביטוי זה (שעליכם להמציא) יופיעו לפחות שלושה מופעים של **set**, לפחות שלושה מופעים של **get** ולפחות שלוש פעולות חשבון.

2

אוניברסיטת
אריאל
בשומרון

שאלה 2 — המימוש של ה-interpreter במודל הסביבות ובמודל ההחלפות — (15 נקודות):

לצורך פתרון שאלה זו מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל ה-substitution והשני במודל הסביבות.

לפניכם מספר שאלות פשוטות. ענו תשובות קצרות (שלוש שורות לכל היותר).

סעיף א' (5 נקודות):

מה מייצג הטיפוס FLANG? באיזה חלק של תהליך האינטרפרטציה הוא משמש אותנו (מה השתנה בנושא זה בין שני המימושים של האינטרפרטר -- הראשון במודל ה-substitution והשני במודל הסביבות) .

סעיף ב' (5 נקודות):

תהליך האינטרפרטציה בשני המודלים שונה בפונקציה eval בטיפול בבנאי Id. הסבירו מה מתבצע במקרה זה בכל אחד מהמימושים ומדוע.

סעיף ג' (5 נקודות):

מהו ההבדל המרכזי בתהליך האינטרפרטציה בין שני המודלים? מה הייתה המוטיבציה מאחורי שינוי זה?

{with {x 1} {+ x 1}} ...

אוניברסיטת אריאל בשומרון

שאלה 3 — FLANG — (35 נקודות):

לצורך פתרון שאלה זו מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל ה-substitution והשני במודל הסביבות.

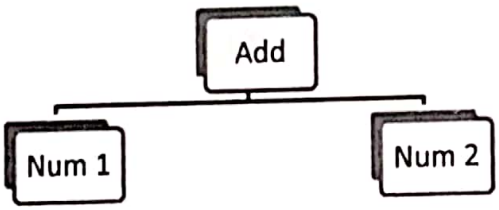
נתון הקוד הבא:

```
(run "{with {Mul-x {fun {x} {* x z}}}
        {with {z 3}
          {call Mul-x z}}}")
```

סעיף א' (6 נקודות):

ציירו את עץ התחביר האבסטרקטי המתאר את הביטוי הנתון במרכאות (כלומר את התוצאה של הפעלת parse על ביטוי זה).

דוגמא 1: העץ המתאר את הביטוי `{+ 1 2}` הוא:



תאור חלופי עבור עץ זה הוא:

(Add (Num 1) (Num 2))

סעיף ב' (12 נקודות):

בתהליך ההערכה של ביטוי זה במודל ה- substitution (על-פי ה-interpreter העליון מבין השניים המצורפים מטה) תופעל הפונקציה eval 10 פעמים. תארו את 10 ההפעלות הללו על-פי סדר הופעתן בחישוב. לכל הפעלה תארו את הפרמטר האקטואלי עליו מופעלת הפונקציה eval וכן את הערך המוחזר מהפעלה זו. הסבירו בקצרה כל מעבר.

דוגמא 2:

בתהליך החישוב של העץ מדוגמא 1, יתבצעו ההפעלות הבאות של eval (מימין מופיעות תוצאות החישוב).

1) (eval (Add (Num 1) (Num 2))) => (Num 3)
2) (eval (Num 1)) => (Num 1)
3) (eval (Num 1)) => (Num 2)

כדי להציג את תשובתכם באופן נוח, כתבו מהם Res_i , Arg_i עבור $i = 1$ to 10, באשר Arg_i מייצג את הפרמטר הפורמלי בקריאה ה-i ל-eval ו- Res_i מייצג את הערך המוחזר מהפעלה זו.

סעיף ג' (12 נקודות):

בתהליך ההערכה של ביטוי זה במודל ה- environment (על-פי ה-interpreter התחתון מבין השניים המצורפים מטה) תופעל הפונקציה eval 10 פעמים. תארו את 10 ההפעלות הללו על-פי סדר הופעתן בחישוב. לכל הפעלה תארו את שני הפרמטרים האקטואלים עליהם מופעלת הפונקציה eval וכן את הערך המוחזר מהפעלה זו. הסבירו בקצרה כל מעבר.

דוגמא 3:

בתהליך החישוב של העץ מדוגמא 1, יתבצעו ההפעלות הבאות של eval (מימין מופיעות תוצאות החישוב).

```
1)  (eval (Add (Num 1) (Num 2))
          (EmptyEnv))                  => (NumV 3)
2)  (eval (Num 1)
          (EmptyEnv))                  => (NumV 1)
3)  (eval (Num 2)
          (EmptyEnv))                  => (NumV 2)
```

סעיף ד' (5 נקודות):

הסבירו ממה נובע ההבדל בתוצאת החישוב בשני בתהליכי ההערכה של הביטוי זה במודל ה- substitution ובמודל ה- environment. מהי התוצאה הרצויה מבחינתנו? הסבירו.

שאלה 4 — הרחבת השפה — (29 נקודות):

לצורך פתרון שאלה זו שוב נעזר בקוד ה- interpreter של FLANG במודל ה substitution, המופיע בסוף טופס המבחן (העליון מבין השניים המופיעים שם).

נרצה להרחיב את השפה ולאפשר מציאת סכום הריבועים מ-0 ועד הפרמטר n (כולל).

להלן דוגמאות לטסטים שאמורים לעבוד:

```
(test (run "{sumsqrbelow 2}") => 5)
(test (run "{+ 2 {sumsqrbelow {+ 3 2}}}") => 57)
(test (run "{call {fun {x}
                {+ {sumsqrbelow x} 1}}
                4}")        => 31)
```

$0^2+1^2+2^2 = 1+4 = 5$

$2+0^2+1^2+2^2+3^2+4^2+5^2 = 2+1+4+9+16+25 = 57$

$1+0^2+1^2+2^2+3^2+4^2 = 1+1+4+9+16 = 31$

סעיף א' (10 נקודות):

ראשית נגדיר פונקציה בשפה pl (הגרסה של Racket בה אנו משתמשים). שורת ההכרזה על הפונקציה תהיה

(: sum-square-below : Number -> Number)

ניתן להניח (ואין צורך בבדיקת נכונות) שהקלט x הוא מספר טבעי (יכול להיות 0). הפונקציה תחשב את סכום **הריבועים** של הטבעיים x...0. כתבו את הפונקציה כך שכל הקריאות הרקורסיביות הן קריאות זנב.

להלן דוגמאות לטסטים שאמורים לעבוד:

```
(test (sum-square-below 1) => 1)
(test (sum-square-below 2) => 5)
(test (sum-square-below 3) => 14)
```

סעיף ב' (2 נקודות):

הרחיבו את הדקדוק בהתאם (הוסיפו את הקוד הנדרש היכן שכתוב –«fill-in»–):

```
#|  The grammar:
    <FLANG> ::= <num>
              | { + <FLANG> <FLANG> }
              | { - <FLANG> <FLANG> }
              | { * <FLANG> <FLANG> }
              | { / <FLANG> <FLANG> }
              | { with { <id> <FLANG> } <FLANG> }
              | <id>
              | { fun { <id> } <FLANG> }
              | { call <FLANG> <FLANG> }
                  | { –«fill-in»–} ; Add
|#
```

סעיף ג' (2 נקודות):

הוסיפו את הקוד הנדרש (היכן שכתוב –«fill-in»–) ל –

```
(define-type FLANG
... ראו קוד ה- interpreter מטה      ...
    [Ssb –«fill-in»– ]) ; Add
```

סעיף ד' (3 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: parse-sexpr : Sexpr -> FLANG)
  ;; to convert s-expressions into FLANGs
  (define (parse-sexpr sexpr)
    (match sexpr
... ראו קוד ה- interpreter מטה      ...
      [–«fill-in»–]     ; Add
      [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))
```

סעיף ה' (4 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: subst : FLANG Symbol FLANG -> FLANG)
  ;; substitutes the second argument with the third argument in the
  ;; first argument, as per the rules of substitution; the resulting
  ;; expression contains no free instances of the second argument
  (define (subst expr from to)
    (cases expr
... מטה interpreter -ראו קוד ה                    ...

      [—«fill-in»—]))  ; Add
```

סעיף ו' (8 נקודות):

עתה נרצה לאפשר לפונקציה eval לטפל במקרה שנוסף עבור פעולת sumsqrbelow.

הערה: אינכם צריכים לטפל במקרה שבו ערך הביטוי עליו מופעלת sumsqrbelow הינו מספר לא טבעי. אולם, עליכם עדיין לוודא שאינו פונקציה.

לצורך כך נגדיר פונקצית עזר: **הוסיפו את הקוד הנדרש במקום המתאים**

```
(: ssb-op : FLANG -> FLANG)
  (define (ssb-op expr)
    (: Num->number : FLANG -> Number)
    (define (Num->number e)
      (cases e
        [(Num n)  n]
        [else (error 'ssb-op "expects a number, got: ~s" e)]))
    (Num —«fill-in»—))   ; Add
```

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: eval : FLANG -> FLANG)
  ;; evaluates FLANG expressions by reducing them to *expressions*
  (define (eval expr)
    (cases expr
... מטה interpreter -ראו קוד ה                    ...

      [—«fill-in»—]))   ; Add
```

הדרכה: השתמשו בפונקציה שלכם מסעיף א'.

מ' - 14/6/5 , סמסטר ב' - מועד א'.

שאלה 1:

(א). ambiguity - קיומה של טעות שניתן לצאר אותה לפי שני עצי צזירה שונים.

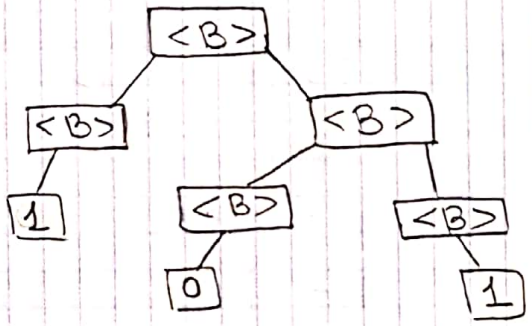
פושמאו לדקדוק אשר סובל מ- ambiguity -

```
<B>::= 0
     | 1
     | <B> <B>
```

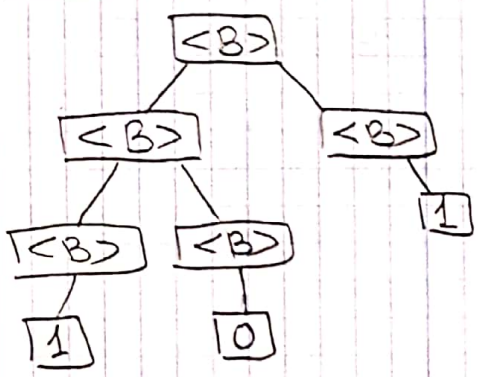
פושמאו למה הדקדוק סובל מהבדו- משמעות -

101

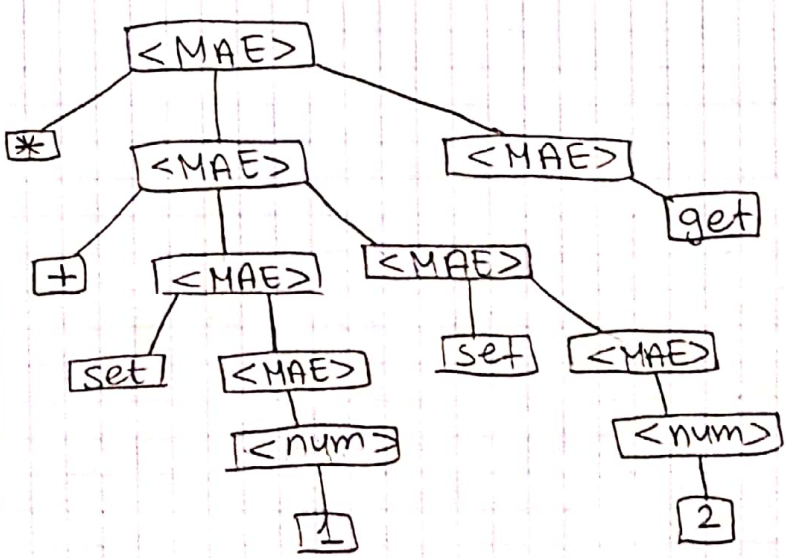
עץ 1 -



עץ 2 -



(ב). צזירה של הטקסט מהדקדוק המצוג -



- הבעיה שמופיעה בביטוי זה היא ש- get - הינו שליפת ערך מה מילון (בפה שלה נשמר ערך יחיד במילון) ובפונקציה הנתונה נשמרו באותו שלב 2 - ערכים במילון - ישנם 2 - set - ים באותו שלב, עם אותן לא יופיעים מה הערך שישלף מהמילון כאשר נערך את הביטוי

(ב) נכתוב פקודות תקין לשפה.

<MAE>::=(1){Seq <AE>}

(2) |{Seq {set <AE>} {Set <AEG>}...<AEG>}

(*) ... זה אומר מופעים של הביטוי שיש לפני ה- ...

<AE>::=(3)<num>

(4) |{+ <AE> <AE>}

(5) |{* <AE> <AE>}

(6) |{/ <AE> <AE>}

(7) |{- <AE> <AE>}

<AEG>::=(8)<num>

(9) |get

(10) |{+ <AEG> <AEG>}

(11) |{- <AEG> <AEG>}

(12) |{* <AEG> <AEG>}

(13) |{/ <AEG> <AEG>}

(ג) הביטוי שיצטר -

"{Seq {set {+ 8 2}}
{Set {- get 10}}
{Set {+ get get}}
8}

נרשום את השלבים לפי הכלל צריך -

<MAE> (2)⇒ {Seq {set <AE>} {set <AEG>} {Set <AEG>} <AEG>}

(4)⇒ {Seq {set {+ <AE> <AE>}} {Set <AEG>} {Set <AEG>} <AEG>}

(3)⇒ {Seq {Set {+ <num> <num>}} {Set <AEG>} {Set <AEG>} <AEG>}

(11)+(10)⇒ {Seq {Set {+ <num> <num>}} {Set {- <AEG> <AEG>}} {Set {+ <AEG> <AEG>}} <AEG>}

(9)+(8)

$\Rightarrow$ {Seq {Set {+ <num> <num>}} {Set {- get <num>}}
{Set {+ get get}} <num>}

$\Downarrow$

{Seq {Set {+ 8 2}} {Set {- get 10}}
{Set {+ get get}} 8}.

שאלה 2:

(א). הטיפוס FLANG מייצג - ביטויים אריתמטיים (+, -, /, *)
- ביטוי with.
- פונ' first-class
- קריאה לפונ'

בעזרת substitution (מודל ההחלפה) ה- FLANG משמש אותנו כדרך מוחצנת מה- parse (ניתוח תחבירי), ב- subst - שבו מתבצעת ההחלפה., וב- eval (הערכה) גם כדרך מוחצנת. במודל הסביבה, הוא משמש כדרך מוחצנת מה- parse, וב- eval.

(ב). במודל ההחלפה -substitution- ב- eval מטיפול בבנאי Id מתבצעת החזרה של שגיאה. משום שאנו מגיעים ל- eval לה אומר שכבר בידענו את כל ההחלפות הנדרשות, ואם הגענו ל- eval עם Id - זה אומר שיש לנו משתנה חופשי - יצר שגיאה.

במודל הסביבה - ב- eval מטיפול בבנאי Id. מתבצע חיפוש (פונ' lookup) של ה- Id בסביבה שבה אנו נמצאים באותו הרגע (הסביבה שקיבל ה- eval), כדי לבדוק את הערכה ל- Id.

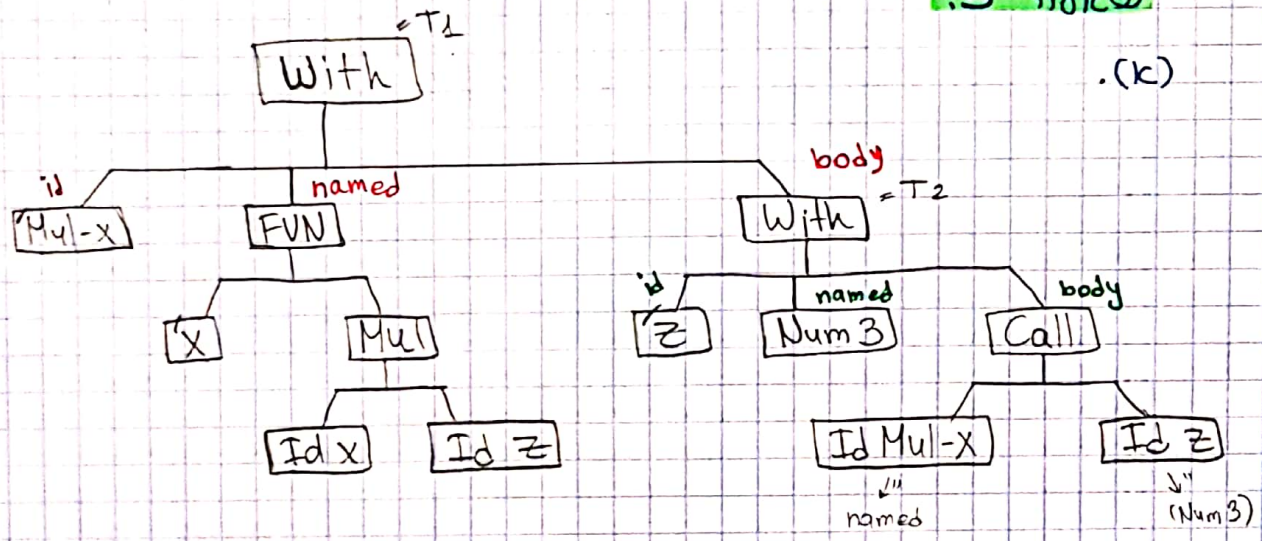
(ג). ההבדל המרכזי בתהליך האינטרפרטציה הוא- שבמודל ההחלפה, אנו נעבור על כל הקוד ונחליף את כל המופעים החופשיים שלנו, ורק לאחר מכן נעשה הערכה לכל הקוד

ובמובן הסביבות אנו כן פעם נשמור את הסביבה שלנו, ובאופן נעשה הפירוק לאותו פיטוי נקרים אותו מסביבה שבה הוא הוגדר.

תחילה המוטיבציה היתה לעבור את מודל ההתפסה (כי לפיהו כן פעם על כל הקלט), והחלטנו ל- SC, כמודל ה- SC בדיוק מודל שמתנהג כמו Dynamic scoping, וכך זאו מה שרצינו, אפשר בפי לתצור ל- static scoping יבטו את מודל הסביבה. ושמרנו על הגדרות שלנו (בהשוואה ל- מודל ההתפסה).

שאלה 3:

(א).



(ב).

Arg1 = T1 - eval על כל הקלט

Res1 = (Num 9)

Arg 2 = (Fun x (Mul (Id x) (Id z))) - eval על ה- named של ה- with (הראשון)

Res 2 = (Fun x (Mul (Id x) (Id z)))

(subst)

Arg 3 = T2 - eval על ה- body של ה- with (הראשון) לאחר ההחלפה של Mul-x'

Res3 = (Num 9)

Arg4 = (Num 3) - eval על ה- named של ה- with השני

Res4 = (Num 3)

(ב.3) .(ק)

Arg5: (Call (Fun x (Mul (Id x) (Num 3))) (Num 3))

Res5: (Num 9)

fun — arg

המשך ה-eval של ה-body ה- with הנשר לאחר ההחלפה של ב')

Arg6: (Fun x (Mul (Id x) (Num 3)))

Res6: (Fun x (Mul (Id x) (Num 3)))

id — body — (Num 3)

לפי eval - ה- fun-expr של ה- call.

Arg7: (Num 3)

Res7: (Num 3)

לפי eval - ה- arg-expr של ה- call. subst

Arg8: (Mul (Num 3) (Num 3)) → לפי eval - ה- body של ה- fun - לאחר ההחלפה.

Res8: (Num 9)

Arg9: (Num 3) →

Res9: (Num 3) — לפי eval - ה- האופרנדים של Mul

arith-op

Arg10: (Num 3) →

Res10: (Num 3)

(ב).

Arg1: T1

Env1: (EmptyEnv)

Res1: Res10

Arg2: (Fun x (Mul (Id x) (Id z)))

Env2: (EmptyEnv)

Res2: (FunV x (Mul (Id x) (Id z)) (EmptyEnv))

Arg3: T2

Env3: (Extend Mul-x (FunV x (Mul (Id x) (Id z)) (EmptyEnv)) (EmptyEnv))

Res3: Res10

Arg 4: (Num 3)
Env 4: Env 3
Res 4: (NumV 3)

Arg 5: (call (Id Mul-x) (Id z))
fun arg
Env 5: (Extend z (NumV 3) (Extend Mul-x
(FunV x (Mul (Id x) (Id z)) (EmptyEnv))
(Empty Env))
Res 5: Res 10.

Arg 6: (Id Mul-x)
Env 6: Env5
Res 6: (FunV x (Mul (Id x) (Id z)) (Empty Env))
id body f-env

Arg 7: (Id z)
Env 7: Env 5
Res 7: (NumV 3)

Arg 8: (Mul (Id x) (Id z))
Env 8: (Extend x (NumV 3) (Empty Env))
Res 8: Res 10

Arg 9: (Id x)
Env 9: Env 8
Res 9: (NumV 3)

Arg 10: (Id z)
Env 10: Env 8
Res 10: error - lookup no binding for "z"

3.(ג) ההבדל בתוצאות החישוב היו - שבמודל ההחלפה היה באג משום שנוצר מצב שבהגדרה של הפונ' x-Mul - היו משתנים חופשיים שלפי אותו scope אז היה להם הגדרה, שמשום שתחילה בידענו את ההחלפה המודל לא יודע לדעת שהפונ' לא הוגדרה באמת scope. במודל הסביבתי אנו תמיד שומרים את הסביבה של הפונ' ראשי מגדירים אותה.

לכן, התאבה הרצויה היא - השגיאה, כלומר התשובה שקיבלנו במודל הסביבתי.

מכיוון ששפת המודלים מתיימרים להתנהגות של static scoping, כלומר לשמור את הסביבה של הפונקציה במקום שבו הוגדרה לכן אנחנו אומרים לאפשר לעשות binding למשתנים שלא הוגדרו במקום הגדרת הפונקציה, ובמודל ההחלפה כן יבוצע binding כזה, לכן התוצאה של מודל ההחלפה שגויה.

שאלה 4:

(א)
```
(: sum-square-below : Number -> Number)
(define (sum-square-below n)
    (sum_help n 0))

(: sum_help : Number Number -> Number)
(define (sum_help n acc)
  (if (eq? n 0) acc
      (sum_help (- n 1) (+ acc (* n n)))))
```

4. (p). | { sumsqrbelow <FLANG>}

(t), [Ssb FLANG]

(7). [(list 'sumsqr below hs)
(Ssb (parse-sexpr hs))]

(7). [(Ssb n) (Ssb (subst n from to))]

(1).(1) (Num (sum-square-below (Num-> number expr)))
(2) [(Ssb n) (ssb-op (eval n))]