הפיס

מבחן 14.06.2015 מועד א':

שאלה 1: BNF

א. הרחבת שפת FLANG לאופרטורים

<FLANG>::=
rule 10 | { not <FLANG> }
rule 11 | { > <FLANG> <FLANG> }
rule 12 | { < <FLANG> <FLANG> }
rule 13 | { = <FLANG> <FLANG> }
rule 14 | { if <FLANG> {then do <FLANG>} {else do <FLANG>} }
rule 15 | <True>
rule 16 | <False>

ב. עבור כל אחת מהמילים הבאות קבעו אם ניתן לייצר את המילה מהדקדוק שייצרנו בסעיף א', אם התשובה כן הראו תהליך גזירה

1) #f - לא, מכיוון שלא ניתן לגזור מילה מהסוג הזה בהקדוק החדש שכתבנו ל-FLANG, מכיוון ש- FLANG לא מכיר אותו אלא FLASE

2) { + 1 22 } כן, ניתן לגזור מילה כזו

<FLANG> $\overset{2}{\Rightarrow}$ { + <FLANG> <FLANG> }
$\overset{1}{\Rightarrow}$ { + <Num> <FLANG> }
$\overset{1}{\Rightarrow}$ { + <Num> <Num> }
סה"כ התוצאה => { + 1 22 }

3) { < 2 True } כן, ניתן לגזור מילה כזו

<FLANG> $\overset{12}{\Rightarrow}$ { < <FLANG> <FLANG> }
$\overset{1}{\Rightarrow}$ { < <Num> <FLANG> }
$\overset{15}{\Rightarrow}$ { < <Num> <True> }
סה"כ התוצאה => { < 2 True }
אם מאוחר יותר נראה כשהר הביטוי כבר עבר ל- eval נקבל ארור

4) { fun {x} {< x x} } כן, ניתן לגזור מילה כזו

<FLANG> $\overset{8}{\Rightarrow}$ {fun {<id>} <FLANG>}
$\overset{12}{\Rightarrow}$ {fun {x} {< <FLANG> <FLANG>}}
$\overset{8}{\Rightarrow}$ {fun {x} {< <id> <FLANG>}}
$\overset{8}{\Rightarrow}$ {fun {x} {< <id> <id>}}
סה"כ התוצאה => {fun {x} {< x x}}

5) { with { foo { fun {x}
{if {< x 2>} {then-do x} {else-do {/ x 2}}}}}
foo}

<FLANG> $\overset{6}{\Longrightarrow}$ { with {<id> <FLANG>} <FLANG>}

$\overset{8}{\Longrightarrow}$ { with { foo { fun {<id>} <FLANG>}} <FLANG>}

$\overset{14}{\Longrightarrow}$ { with { foo { fun {x}
{if <FLANG> {then-do <FLANG>} {else-do <FLANG>}}}}
<FLANG>}

$\overset{12}{\Longrightarrow}$ { with { foo { fun {x}
{if {< <FLANG> <FLANG>>}
{then-do <FLANG>} {else-do <FLANG>}}}}
<FLANG>}

$\overset{7,1}{\Longrightarrow}$ { with { foo { fun {x}
{if {< <id> <Num>}
{then-do <FLANG>} {else-do <FLANG>}}}}
<FLANG>}

$\overset{7}{\Longrightarrow}$ { with { foo { fun {x}
{if {< x 2}}
{then-do <id>} {else-do <FLANG>}}}}
<FLANG>}

$\overset{5}{\Longrightarrow}$ { with { foo { fun {x}
{if {< x 2}}
{then-do x} {else-do {/ <FLANG> <FLANG>}}}}}
<FLANG>}

$\overset{7,1}{\Longrightarrow}$ { with { foo { fun {x}
{if {< x 2}}
{then-do x} {else-do {/ <id> <Num>}}}}}
<FLANG>}

$\overset{7}{\Longrightarrow}$ { with { foo { fun {x}
if {< x 2}}
{then-do x} {else-do {/ x 2}}}}}} foo}

⑥ {call x y}

כן, ניתן לגזור

<FLANG> $\overset{⑥}{\Longrightarrow}$ {call <FLANG> <FLANG>}

$\overset{④}{\Longrightarrow}$ {call <id> <FLANG>}

$\overset{④}{\Longrightarrow}$ {call <id> <id>}

לפי הנתונים => {call x y}

שאלה ב: שאלות כלליות

א. איזו מהמשפטים נכונים לגבי dynamic vs. static scoping?

תשובה ב': שימוש בדינמי מאפשר שבמקרה של תוכנה לנתוקל בנק שמקבלת כפרמטר סכום הדושרה שהתחלפו בנק הראון הטו שפרמטר לה משתנה בכל יום ובכן אפשרון מקומי שלו ישנה את הפקוד התוכנה כולה.

I. איזו מהמשפטים הבאים נכונים לגבי מהלך passing?

תשובה א': לפחות מהלך ה- passing פרוק להבדלות אם כאשר הקוד אינו תקין

א. איזו מהמשפטים הבאים נכונים לגבי שפות המתיחסות לפונקציות first class?

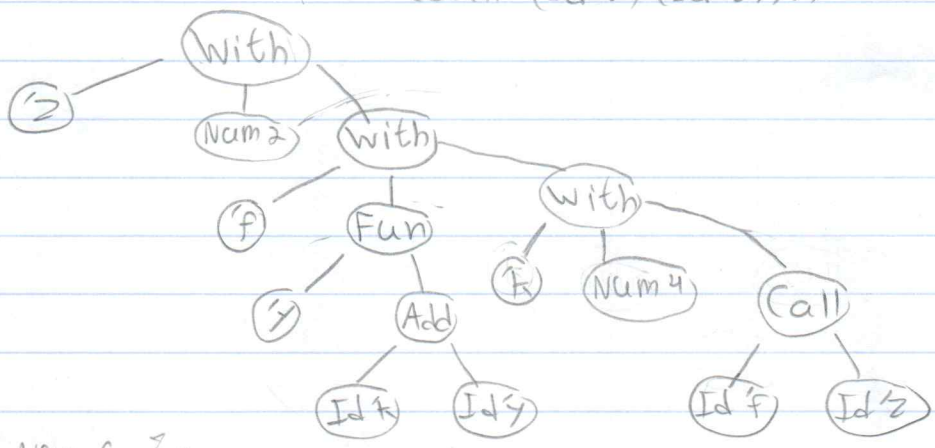
שאלה 3: הרצה של test

נתון הקוד הבא:

```
(run "{with {z 2}
        {with {f {fun {y} {+ {x y}}}}
              {with {x 4}
                    {call f z}}}}")
```

א. תארו את פעולת ה- eval בתהליך הערכה של הקוד במודל ההתלפות

```
AST1: (With 'z (Num 2) (With 'f (Fun 'y (Add (Id x) (Id y)))
                           (with 'x (Num 4)
                                 (call (Id 'f) (Id 'z))))
```



RET1: Num 6 כמו

AST2: (Num 2)

RET2: (Num 2)

AST3: (with 'f (Fun 'y (Add (Id x) (Id y)))
(with 'x (Num 4) (call (Id 'f) (Id 2))))

RET3: Num 6 כמו

AST4: (Fun 'y (Add (Id x) (Id y))

RET4: (Fun 'y (Add (Id x) (Id y)) בגלל subst

AST5: (with 'x (Num 4) (call (Fun 'y (Add (Id 'x) (Id 'y)) (Num 2)))

RET5: Num 6 כמו

AST6: (Num 4)

RET6: (Num 4)

AST7: (call (Fun 'y (Add (Num 4) (Id 'y)) (Num 2)))

RET7: Num 6 כמו

AST8: (Fun 'y (Add (Num 4) (Id 'y)) (Num 2)

RET8: " " " " " "

AST_9: (Num 2)

RET_9: (Num 2)

AST_{10}: (Add (Num 4) (Num 2))

RET_{10}: Num 6 מוחזר בסוף

AST_{11}: Num 4

RET_{11}: Num 4

AST_{12}: Num 2

RET_{12}: Num 2

תשובה סופית יוחזר Num 6

I_ מעקב הסבר בתוך

AST1: (with 'z (Num 2) (with f (Fun 'y (Add (Id 'k) (Id f))) (with 'k (Num 4) (call (Id f) (Id 'z))))) — id: 'z, name: (Num 2), body: (with f (Fun 'y (Add (Id 'k) (Id f))) (with 'k (Num 4) (call (Id f) (Id 'z)))))

RET1:

ENV1: (EmptyEnV)

AST2: (Num 2)

RET2: (NumV 2)

ENV2: (EmptyEnV)

AST3: (with 'f (Fun 'y (Add (Id 'k) (Id 'f))) (with 'k (Num 4) (call (Id 'f) (Id 'z)))) — id: 'f, name: (Fun 'y (Add (Id 'k) (Id 'f))), body: (with 'k (Num 4) (call (Id 'f) (Id 'z)))

RET3:

ENV3: (Extend 'z (NumV 2) (EmptyEnV))

AST4: (Fun 'y (Add (Id 'k) (Id 'f))) — id: 'y, body: (Add (Id 'k) (Id 'f))

RET4: (FunV 'y (Add (Id 'k) (Id 'f)) ENV3)

ENV4: ENV3

AST5: (with 'k (Num 4) (call (Id 'f) (Id 'z))) — id: 'k, name: (Num 4), body: (call (Id 'f) (Id 'z))

RET5:

ENV5: (Extend 'f (FunV 'y (Add (Id 'k) (Id 'f)) ENV3) ENV3

AST6: (Num 4)

RET6: (NumV 4)

ENV6: ENV5

AST7: (Call (Id 'f) (Id 'z)) — fun: (Id 'f), arg: (Id 'z)

RET7:

ENV7: (Extend 'k (NumV 4) ENV5), ENV6)

AST8: (Id 'f)

RET8: (FunV 'y (Add (Id 'k) (Id 'f)) ENV3 -> lookup — id: 'y, body: (Add (Id 'k) (Id 'f))

ENV8: ENV7

AST9: (Id 'z)

RET9: (NumV 2)

ENV9: ENV8

AST10: (Add (Id 'k) (Id 'f))

RET10:

ENV_{10}: Extend 'x (Num 2), ENV_3)

AST_{11}: (Id 'x)

RET_{11}: Error -- ...

לאחר שנעשה השאלה הזו נחזור שגיאה מכיון שה- Fun שנשלח לו לא מכיל את x.

X. למה יש הבדל בין שני ההרצות.

מכיוון שבמודל ההחלפות קיים באג שמגיע ל- dynamic scope, מה שלא קורה במודל הסביבות כיון שהוא מקודד לפי lexical מה מעומר על ה- scope תקין לפרמטרים הסיבה לכך היא שבמודל הסביבות אנחנו מוסיפים לפונקציה אובייקט סביבה מתוחכם הרלוונטית לו, בעוד שבשני מנסים שאלה ל- Id תוצרת שגיאה כיון של- x אין הגדרה מתאימה, לעומת זאת במודל החלפות בגרסת static והחלפנו את המספר על x בלי הקשר של סביבה.

בקיצור- התוצאה הרצויה היא שגיאה, הבעיה היא רק שבמודל הסביבות אנחנו צריכים סביבה הרלוונטית בעל ה- scope הנכון שצריך מהות ההבדל בין הגרסאות לפונקציות scope

המשך eval ←

```
[[IF l m r]
  (let [fval (eval l)]
   cases fval
   [(Bool true) (eval m)]
   [else (eval r)] )]
```

7. נשתמש בפונקציות הבאות

```
(: run : string -> (U Number Boolean FLANG)
  (define (run str)
    (let ([result (eval (parse std))])
      (cases result
       [(Bool b) b]
       [(Num n) n]
       [(Fun ....) result]
       [else....
```

בלבד

שאלה 4: הרחבת שפה

א. הרחבת טיפוס FLANG

```
(define-type FLANG)
  [Num Number]
   ⋮
  [Bool (U true false)]
  [Bigger FLANG FLANG]
  [smaller FLANG FLANG]
  [Equal FLANG FLANG]
  [Not   FLANG]
  [If <FLANG> <FLANG> <FLANG>
```

ב. parse

```
(define (parse-sexpr sexpr)
 (match sexpr
   ⋮
  [`true (Bool true)]
  [`false (Bool false)]
   ⋮
  [(list '= l r) (Equal (parse-sexpr l) (parse-sexpr r))]
  [(list '> l r) (Bigger (parse-sexpr l) (parse-sexpr r))]
  [(list '< l r) (smaller (parse-sexpr l) (parse-sexpr r))]
  [(list 'not exp) (not (parse-sexpr exp))]
  [(list 'if (list condition (`then-do ex1) (`else-do ex2)))
        (If (parse-expr condition)
            (parse-expr ex1) (parse-expr ex2))]
 [else...
```

subst .x

```
(define (subst expr from to)
  [(Bool b) expr]
  [(Equal l r) (Equal (subst l from to) (subst r from to))]
  [(Bigger l r) (Bigger (subst l from to) (subst r from to))]
  [(Smaller l r) (Smaller (subst l from to) (subst r from to))]
  [(Not b) (Not (subst b from to))]
  [(If con ex1 ex2) (If (subst con from to)
                        (subst ex1 from to)
                        (subst ex2 from to))]
```

ד. פונקציות עזר

1) פונקציה בולאנית פו מקומית המתעדכנת FLANG

```
(: logic-op: (Number Number -> Boolean) FLANG FLANG -> FLANG)
(define (logic-op op expr1 expr2)
  (Bool (op (Num->number expr1) (Num->number expr2))))
```

2) מקבלת FLANG ומחזירה את הערך הבוליאני המתאים

```
(define (FLANG->Boo e)
  (cases e)
  [(Bool b) (if (eq? b 'false) #F #t)]
  [else (error 'arith-op "expects a boolean: ~s" e)]))
```

ה. טיפול פונקציית eval לפי התאכות

```
(: eval: FLANG -> FLANG)
(define (eval expr
  (cases expr
  [(Bool b) (FLANG->Boo expr)]
  [(Equal l r) (logic-op '= (eval l) (eval r))]
  [(Bigger l r) (logic-op '> (eval l) (eval r))]
  [(Smaller l r) (logic-op '< (eval l) (eval r))]
  [(Not exp) (if (FLANG->bool exp) (FLANG->bool true)
                                   (FLANG->bool false))]
```

<-