.א 1

```
~~<ROL>::={reg-len <num>~~
<ROL>::={reg-len  = <num>  <RegE>}        (1)

                 <Bits><Bits>...
<RegE>:= { <Bits>...}                     (2)
       | {shl  <RegE>}                    (3) ✓
       | {and  <RegE> <RegE>}             (4) ✓
       | {or   <RegE> <RegE>}             (5) ✓
       | {with {<id> <RegE>} <RegE>}      (6) ✓
       | <id>                             (7) ✓
       | {fun  {<id>}  <RegE>}            (8) ✓
       | { call  <RegE>  <RegE>}          (9) ✓
    -> | { if  <BoolE>  <RegE> <RegE>}    (11)
     > |<BoolE>
<Bits> ::= 0    |   1              ✓     (12)
          (10)     (11)
```

10/10

~~| true~~
~~| false~~

Reg f תכלס צריך להוסיף
אבל גם כך בסדר

```
<BoolE>:: = true                          (12) ✓ .ב
          | false                         (13) ✓
          | {geq? <RegE> <RegE>}          (14) ✓
          | {maj  <RegE> }                (15) ✓
```

6/6

ג. המילה:

```
{reg-len = 2 {if {geq? {1 1} {0 0}}
                 {Shl {0 0}}
                 {and {1 1}
                    {or {0 0} {1 1}}}}}
```

הגזירה:

4/4

(1) {reg-len = <num> <RegE>}

<num> = 2

(11) {if <BoolE> <RegE> <RegE>}

(14) {geq? <RegE> <RegE>}

(2) {<Bits>...} = {1 1}

(10) <Bit> = 1

(2) {<Bits>...}

(11) <Bits> = 0

~~<RegE> = {Shl {0 0}}~~

(3) <RegE> = {Shl <RegE}}

(2) {<Bits>...}

(10) <Bits> = 0

(4) {and <RegE> <RegES}

(2) <RegE> = {<Bits>... }

(11) {Bits> = 1

(5) <RegE> = {or <RegE> <BegE>}

(2) <RegE> = {<Bits> ...}

(10) <Bits> = 0

(2) <RegE> = {<Bits>...}

(12) <Bits> = 1

2. א. יתרונות שמות מצהים:

1. במקום לחשב הרבה פעמים ערך שנמצא בכמה מקומות מחשבים אותו פעם אחת ומשתמשים בו - יעיל יותר.

2. המתכנתים פחות טועים בהצהרת הקבוע בכמה מקומות ולכן יכתבו אותו פעם אחת.

3. קריאות - במקום כמה מופעים יהיה שם אחד של החישוב.

יתרונות פונקציות:

1. קריאות - ניתן שם אינדיקטיבי לקטע קוד לדוגמא "פונקציית הצהרות".

2. במידה ומתכנתי קוד מסובך ומצאתי דרך יותר יעילה לעשות אותו, נעדכן רק במקום אחד.

3. אם נכתוב את אותו קוד בכמה מקומות עלולים לטעות.

6/6

2.ב.

```
(: Shift-left : Bit-List -> Bit-List)        (1)
(define (Shift-left bl)
   (append (rest bl) (list (first bl))))
```

Fix! 5/5

```
(: majority? : Bit-List -> Boolea)           (2)
(define (majority? bl)
   (Let ([ones (countOnes 0 bl)])
        (>= ones (- (length bl) ones))))
;; בגלל שיכול להיות מספר אי-זוגי של איברים
;; אני בשוויון בודק אם מספר האחדות גדול ממספר
;; האפסים

(: countOnes : Number Bit-List -> Number)
(define (countOnes acc bl)
  (if (eq? (first bl) 1)
      (countOnes (+ acc 1) (rest bl))
      (countOnes acc (rest bl))))
;; הפונקציה countOnes היא פונק' עזר
;; שסופרת את כמות האחדות בעזרת
;; רקורסיית זנב.
```

מעולה! 5/5

```
(q eq -bitlist? : Bit-List Bit-List                    (3)
                         ->Boolean )
(define (qeq-bitlist? bl1 bl2)
  (let ([s1 (length bl1)]
        [s2 (length bl2)])
    (if (= s1 s2) (regEq bl1 bl2)
      (if (> s1 s2)
          (if (eq? (first bl1) 1) true
              (geq-bitlist (rest bl1) bl2))
          (if (eq? (firs bl2) 1) false
              (geq-bitlist bl1 (res bl 2)))))
```

; הרעיון הוא לבדוק אם הגודל שווה, לבדוק רגיל, אם

; אורך הרשימה גדולה יותר לבדוק אם יש אחד

; אז להחזיר תשובה, אחרת להמשיך ברקורסיה

```
(:regEq : Bit-List Bit-List -> Boolean)
(define (regEq bl1 bl2)
  (if (null? bl1) false
    (if (eq? (first bl1) (firs. bl2))
      (regEq (rest bl1) (rest bl2))
      (if (eq? (first bl1) 1)
          true
          false))))
```

היית צריך להוסיף הנחה כי הביטים מסודרים מהביט של החזקה הגבוהה לקטנה

בכל מקרה אחלה פתרון קבל בונוס 2 נקודות.

7/5

.k 3

~~AST_1 = (With '⊥ (Fun 'X (Fun 'y (Mul (Id X) (Id y)))) (With '1~~

AST_1 = (With '⊥ (Fun 'X (Fun 'y (Mul (Id x) (Id y))))
(With 'X (Num 4)
(Call (Call (Id ⊥) (Add (Num 1) (Num 2)))
(Num 6))))

$Cache_1$ = '()

Res_1 = (Num 10)

AST_2 = (Fun 'X (Fun 'y (Mul (Id x) (Id y))))

$Cache_2$ = '()

Res_2 = AST_2

AST_3 = (With 'X (Num 4) (Call (Call (Id ⊥)
(Add (Num 1) (Num 2))) (Num 6)))

$Cache_3$ = '([redacted] '('⊥ AST_2) '())

Res_3 = (Num 10)

[redacted] AST_4 = Res_4 = (Num 4)

$Cache_4$ = $Cache_3$

AST_5 = (Call (Call (Id ⊥) (Add (Num 1) (Num 2)))
(Num 6)

$Cache_5$ = '('('X (Num 4)) $Cache_3$)

Res_3 = (Num 10)

AST_6 = (Call (Id ⊥) (Add (Num 1) (Num 2)))

$Cache_6$ = Cache 5

Res_6 = (Fun 'y (Mul (Id x) (Id y)))

AST_7 = (Id ⊥)

$Cache_7$ = $Cache_5$

Res_7 = (Fun 'X (Fun 'y (Mul (Id x) (Id y))))

AST_8 = (Add (Num 1) (Num 2))

$Cache_8$ = $Cache_5$

Res_8 = (Num 3)

AST_9 = (Num 1) = Res_9

$Cache_9$ = $Cache_5$

AST_{10} = Res_{10} = (Num 2)

$Cache_{10}$ = Cache 5

AST_{11} = (Fun 'y (Mul (Id x) (Id y)))

$Cache_{11}$ = '('('x (Num 3) $Caches_5$)

Res_{11} = AST_{11}

AST_{12} = (Num 6) = Res_{12}

$Cache_{12}$ = $Cache_5$ ✓

AST_{13} = (Mul (Id x) (Id y)

$Cache_{13}$ = '('('y (Num 6)) $cache_5$) ✓

Res_{13} = (Num 10)

15/15

AST_{14} = (Id x)

$Cache_{14}$ = $Cache_{13}$

Res_{14} = (Num 4)

כל הכבוד

AST_{15} = (Id y)

$Cache_{15}$ = $Cache_{13}$

Res_{15} = (Num 6)

לא קיים

AST_{16} = (Num 10) = Res_{16}

$Cache_{16}$ = $Cache_3$

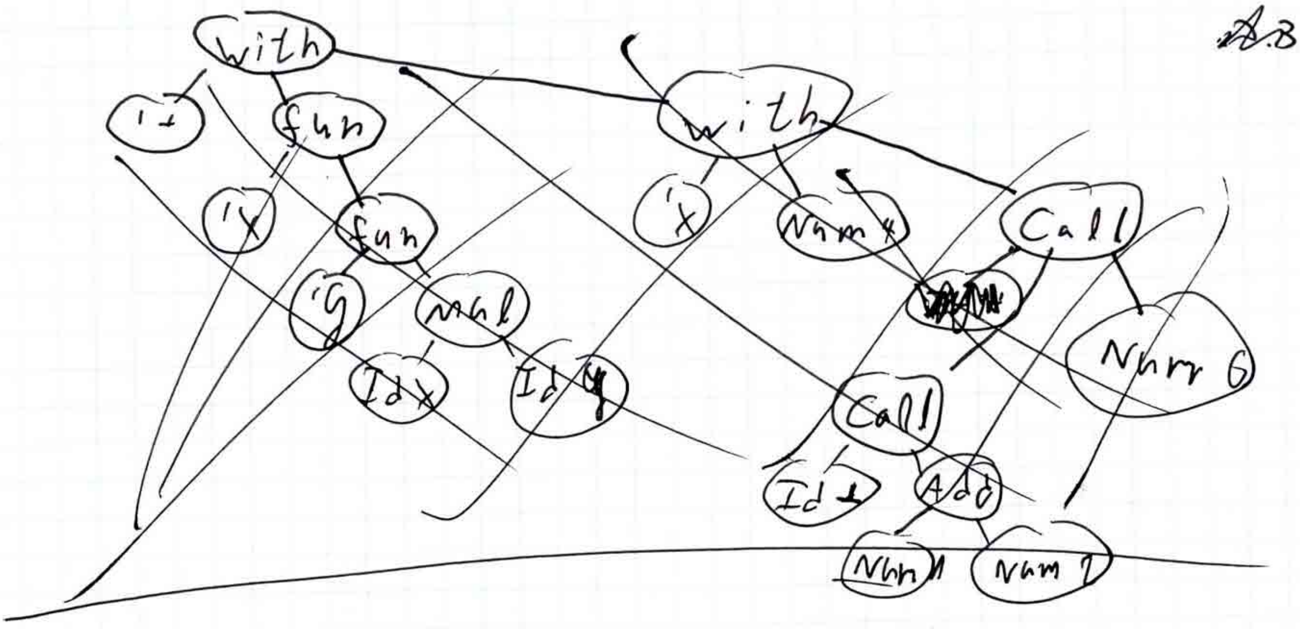
~~התוצאה הסופית היא (Num 10)~~

AST_{17} = (Num 10) = Res_{17}

$Cache_{17}$ = '()

התוצאה הסופית היא (Num 10)

א.3



~~3.ב. [illegible]~~

~~3.ב. כאשר היינו מפעילים את ה eval [illegible] lookup [illegible] הפונקציה עם הארגומנט x [illegible] בסביבה [illegible] ולאחר מכן [illegible]~~

3. ב. היינו סוגרים את הפונקציה עם הארגומנט x בסביבה עם (num 3) ולאחר מכן היינו מפעילים mul ולכן היה חוזר (Num 18).

ג. א, ב, ג

טון לא פספס

6/8

5/7

8 4.א.

```
(define-type RegE
   [Reg  Bit-List]
   [And  RegE  RegE]
   [Or   RegE  RegE]
   [Shl  RegE]
   [Id   Symbol]
  [With  Symbol RegE  RegE]
  [Fun   Symbol RegE]
  [Call  RegE   RegE]
  [Bool  Boolean]
  [Geq   RegE  RegE]
  [Maj   RegE]
  [If  RegE  RegE  RegE])
```

14.5 ב.

```
<fill 1> = (< len 1)
<fill 2> = 'Parse-sexpr "Register length should
              be greater then One Zero"
fill 3 = (Parse-sexpr-RegL reg-sexpr len)
fill 4 = (eq? (length sexpr) reg-len)
fill 5 = (Reg (list->bit-list sexpr))
fill 6 = 'Parse-Sexp-RegL "Given Register size
             does not fit actuall size"
fill 7 = (Bool #t)
fill 8 = ['false (Bool #f)]
fill 9 = (Id name)
                                                   reg-len
fill 10 = (With name (Parsed-sexpr-RegL named)
              (Parsed-sexp-RegL body))
fill 11 = (And (Parse-sexp-RegL lreg)
               (Parse-Sexp-RegL rreg))
```

```
fill 12 = [(list 'or lreg rreg)
             (Or (Parse-sexpr-RegL lreg)
                 (Parse-sexpr-RegL rreg))]
fill 13 = [(list 'Shl reg) (Shl (Parse-sexp-RegL reg))]
fill 14 = (Fun name (Parse-sexp-RegL body))
fill 15 = (Parse-sexpr-RegL fun) (Parse-sexp-RegL arg)
fill 16 = boolE trueE falseE
fill 17 = (If (Parse-sexpr-RegE boolE)
              (Parse-sexpr-RegE trueE)
              (Parse-sexpr-RegE falseE))
fill 18 = (list 'geq? lreg rreg)
            (Geq (parse-sexpr-RegL lreg) (Parse-sexp-RegL rreg))
fill 19 = (list 'maj? reg) (Maj (Parse-sexp-RegL reg))
```

16.5 ג.

```
(define-type VAL
    [RegV Bit-List]
    [FunV symbol RegE ENV]
    [BoolV Boolean])
```

~~fill 3 = (null? bl1)~~
~~fill 4 = (OP (first bl1) (first bl2))~~
~~(reg-arith-op (rest bl1) (rest bl 2))~~
~~fill 5 = (reg-arith-op (RegV->bit-list~~

```
fill 3 = (null? bl 1)
fill 4 = (OP (first bl 1) (first bl2))
           (reg-arith-op (rest bl 1) (rest bl 2)
fill 5 = (reg-arith-op (Reg->bit-list reg1)
                       (Reg->bit-list reg2))
```

```
fill 6 = (RegV n)
fill 7 = (BoolV b)
fill 8 = (reg-arith-op bit-and
                (eval l env) (eval r env))
fill 4 = (reg-arith-op bit-or
               (eval l env) (eval r env))
                           RegV->bit-list
fill 10 = (RegV (Shift-left (eval reg env)))
fill 11 = (lookup name env)
fill 12 = (FunV bound-id bound-body env)
fill 13 = (FunV id body env)
          (eval body (Extend id (eval arg-exp env)
                                   env))
fill 14 = (eval cond-term env)
fill 15 = (cases condVal
             [(BoolV b)
               (if (b) (eval do-term env)
                          (eval else-term env))]
לא את צריך   [else (error 'eva "expected boolean
     כלל               got ~s" condval])
fill 16 = (BoolV (majority? (Reg->bit-list
                       (eval reg env))))

fill 17 = (BoolV (geq-bitlist?
                (Reg->bit-list (eval reg1 env))
                (Reg->bit-list (eval reg2 env))))
```

39/40

אליניר דיגיטל
עוצמת הדפוס הדיגיטלי
03-6240810