גיליון תשובות מספר נבחן:_____

על תשובות ריקות יינתן 20% מהניקוד!

שאלה 1 [35 נקודות]

1.1 [5 נקודות]

```
(for n 0 1) (/ 1 n))
```

1.2 [16 נקודות]

1.2.1

מבנה תחבירי המאפשרת לתאר בצורה פשוטה ביטוי הניתן לתיאור ע"י מבנה אחר בשפה.
לדוגמא: cond - if, let - lambda application

1.2.2

ההמרה דורשת חישוב של high ו low על קריאה ל applicativeEval. כלומר, היא אינה רק שינוי 'בזמן קומפילציה' של המבנה התחבירי (ה AST) אלא דורשת קריאה לאינטרפרטר

1.2.3

```
 | (for <var-decl> <num-exp> <num-exp> <cexp>) // for-exp(var:var-decl,
                                                 low:num-exp, high:num-exp,
body:cexp)
```

1.2.4

```
const For2Apps = (exp: ForExp, env: Env): AppExp[] => {
    const ret: AppExp[] = [];
    const low = exp.low.val;
    const high = exp.high.val;
    for (let i = low; i <= high; i++) {
        ret.push(makeAppExp(makeProcExp([exp.var], [exp.body]),
                                [makeNumExp(i)]));
    }
    return ret;
}
```

1.3 [14 נקודות]

1.3.1

יש לשמור את ה frame עבור כל קריאה רקורסיבית, לשם עיבוד התוצאה אחר כך, כך שעשויה להיווצר בעיית זיכרון עבור מספר רב של קריאות רקורסיביות.

1.3.2

```
(define fact$
  (lambda (n cont)
    (if (= n 1)
        (cont 1)
        (fact$ (- n 1)
                (lambda (res) (cont (* n res)))))))


(define ret 1)
(define fact-for-set
  (lambda (n)
     (set! ret 1)
     (for (i 2 n) (set! ret (* ret i)))
     ret
  )
)
```

1.3.3

חסרון המימוש עם רקורסיית זנב: נדרש לממש באינטרפרטר את האופטימיזציה בה קריאה לרקורסיית זנב אינה משאירה את ה frame פתוח.

חסרון המימוש עם for ו set!: תכנות לא פונקציונאלי, על כל חסרונותיו (הוכחת נכונות, מקביליות, ועוד)

שאלה 2: טיפוסים [25 נקודות]

type unifiers 2.1
[6 נק]

2.1.1

```
[S * [Number -> S1] -> S]
[Pair(T1) * [T1 -> T1]-> T2]
```

Unifier:

{S = Pair(Number), T1 = Number, S1 = Number, T2 = Pair(Number)}

2.1.2

```
[S * [Number -> S] -> S]
[Pair(T1) * [T1 -> T1] -> T2]
```

No unifier:
Constraint 1: S = Pair(T1)
Constraint 2: T1 = Number
Constraint 3: T1 = S -- fails on resolving {Number = Pair(T1)}

2.1.3

```
[T1 * [T1 -> T2] -> N]
[[T3 -> T4] * [T5 -> Number] -> N]
```

Constraints:
T1 = [T3 -> T4]
T1 = T5 = [T3 -> T4]
T2 = Number

Unifier:
{T1 = [T3 -> T4], T2 = Number, T5 = [T3 -> T4]}

Records 2.2

2.2.1 [6 נק]

```
(record ("a" : string) ("b" : number))
```

Represents the set of values:
({"a"} x String) x ({"b"} x Number} x { (Keys - {"a", "b"}) ** V }

```
(record ("a" : (record ("x" : number))) ("c" : boolean))
```

Represents the set of values:
({"a"} x ({"x"} x Number) x ((Keys - {"x"}) ** V)) x
({"c"} x {true, false}) x
((Keys - {"a", "c"}) ** V)

Common errors:

- Write ({"a"} x String) U ({"b"} x Number) with U instead of x -- this is wrong because we want values that include both a value for a and a value for b
- Write ({"a"} ** String) instead of ({"a"} x String) -- this is wrong because it accepts cases where the key "a" does not appear
- Forget the part of "all the other keys can be any value" (that is - answer without the part "x ((Keys - { …. }) x V)"
- Forget to remove the fields "a" and "b" in the "all the other keys can be any value" component (that is, use Keys x V instead of (Keys - {"a", "b"}) x V)
- Write (String ** Number) or (String x Number) for (record (a : String) (b : Number))

2.2.2 [13 נק]

Typing rule Primitive get-record:
For every type environment _Tenv:
For all expressions _r and _k
For all type expression _T
For all string _key
If _Tenv |- _r: (record (_key: _T)),
_Tenv |- _k: {_key}
Then _Tenv |- (get-record _r _k): T

In other words, from the expression (get-record r k) we derive 3 types equations:

1. **r: (record (key: T))**
2. **k: {key} (where key = appEval(k) is a string)**
3. **(get-record r k): T**

```
(lambda (r : Tr) : Tret
      (* 2 (get-record r "b")))
```

- T1: (lambda (r : Tr) : Tproc (* 2 (get-record r "b")))
- T*: *
- T2: 2
- Tget: (get-record r "b")
- Tb: "b"

Type equations:

Expression	Equation
*	T* = (Number * Number -> Number)
2	T2 = Number
"b"	Tb = String
(get-record r "b")	Tr = (record ("b": Tf))
	Tget = Tf
(* 2 (get-record r "b"))	Tget = Number
(lambda (r : Tr) : Tret …)	T1 = (Tr -> Tret)
	Tproc = Tret
	Tret = Tget

Solve the equations:

Tf = Number
Tr = (record ("b" : Number))
T1 = ((record ("b": Number)) -> Number)

Solution:

Tret = Number
Tr = (record ("b" : Number))

שאלה 3: CPS ורשימות עצלות [30 נקודות]

3.1 [4 נקודות]

קוד:

```
const sumIter = (n: number, acc: number): number =>
    (n === 0) ? acc :
    sumIter(n - 1, n + acc);

sumIter(1000000);
```

הסבר:

הקוד ממומש עם רקורסיית זנב ולכן היינו מצפים שהקוד ירוץ ללא בעיות. מכיוון שב-Node אין אופטימיזציית זנב, הקריאה ל-sumIter עם מספר גבוה מספיק תקרוס בגלל stack overflow.

טעויות נפוצות:

- קוד בו הקריאה הרקורסיבית אינה בעמדת זנב.
- קוד הבודק האם ביטוי נמצא בעמדת זנב

3.2 [10 נקודות]

```
Contract:
Type: [[T1 * T2 * [T3 -> T4] -> T4] ;; f$
       * List(T1)                   ;; lst1
       * List(T2)                   ;; lst2
       * [List(T4) -> T5]           ;; cont
      -> T5]
Precondition: lst1 and lst2 are of the same length

(define map2$
  (lambda (f$ lst1 lst2 cont)
    (if (empty? lst1)
        (cont '())
        (f$ (car lst1)
            (car lst2)
            (lambda (f-res)
              (map2$ f$
                     (cdr lst1)
                     (cdr lst2)
                     (lambda (map-res)
                       (cont (cons f-res map-res)))))))))
```

טעויות נפוצות:

- בחתימה, הגבלת הטיפוס ל-Number בלבד
- אופרטור שאינו ב-CPS
- קריאות ל-f$ ול-map2$ שאינן בעמדת זנב
- בניית continuations לא נכונה

3.3 [8 נקודות]

מקרה בסיס:

```
a-e[(append$ '() lst2 k)] =>* a-e[(k lst2)]
                            = a-e[(k (append '() lst2))]
```

הנחת האינדוקציה:

קיים n כך שלכל רשימה lst1 באורך קטן או שווה n מתקיים:

```
a-e[(append$ lst1 lst2 k)] = a-e[(k (append lst1 lst2))]
```

צעד האינדוקציה:

עבור רשימה lst1 באורך n+1:

```
a-e[(append$ lst1 lst2 k)]
=> a-e[(append$ (cdr lst1) lst2 (lambda (res) (k (cons (car lst1) res))))]
```

האורך של (cdr lst1) הוא n ולכן, לפי הנחת האינדוקציה:

```
=> a-e[((lambda (res) (k (cons (car lst1) res))) (append (cdr lst1) lst2))
=> a-e[(k (cons (car lst1) (append (cdr lst1) lst2)))]
=> a-e[(k (append lst1 lst2))]
```

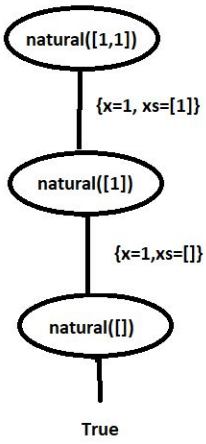
3.4 [8 נקודות]

```
(define fibs
  (letrec ((fibs-gen
             (lambda (a b)
               (cons-lzl a (lambda () (fibs-gen b (+ a b)))))))
    (fibs-gen 0 1)))
```

שאלה 4: תכנות לוגי [20 נקודות]

א.

a. (2 נקודות) ציירו את עץ החישוב עבור השאילתה: ?- natural([1,1])



b. (נקודה) מה סוג העץ המתקבל? האם הוא עץ הצלחה או כישלון סופי או אין סופי?

עץ הצלחה סופי.

c. (2 נקודות) האם היחס מכיל רק מספרים טבעיים בייצוג רשימה? אם כן הסבירו מדוע, אם לא תקנו את היחס.

לא, היחס מכיל גם דברים שאינם מספרים טבעיים לפי הגדרת השאלה (לדוגמא [2,2]).
כדי לתקן את היחס נדרוש ש x=1

```
natural([]).

natural(1|Xs):-natural(Xs).
```

ב.

(5 נקודות) ממשו את היחס equal/2 שמקבל שני מספרים בייצוג רשימות ובודק האם הם שווים.
יש לבדוק שהקלט הוא בייצוג רשימות.

```
% Signature: equal(X, Y)/2.

% Purpose: X =Y.
```

equal([],[]).
equal(X|Xs,Y|Ys):-equal(Xs,Ys).

ג.

a. 5 נק

```
% Signature: fibonacci(N, X)/2.

% Purpose: ____________________________________________


fibonacci([1],[1]).

fibonacci([1,1],[1]).

fibonacci(N1,N2|Ns,F3):-fibonacci(N2|Ns,F1),fibonacci(Ns,F2),
append(F1,F2,F3).
```

b. 5 נק

נגדיר יחס עזר ונשתמש באופרטור _ כדי לא לקבל השמות לX בפתרונות המוצאים.

```
?-even(_X),fibonacci(_X,F).

%signature: even(x)/1
% x modulu 2 =0.

even([]).
even(X1,X2|Xs):-even(Xs).
```