**מבחן בקורס:** עקרונות שפות תכנות, 202-1-2051
**מועד:** ב
**תאריך:** 26/7/2022
**שמות המרצים:** מני אדלר, מיכאל אלחדד, ירון גונן
**מיועד לתלמידי:** מדעי המחשב והנדסת תוכנה, שנה ב', סמסטר ב'
**משך המבחן:** 3 שעות
**חומר עזר:** אסור

**הנחיות כלליות:**

- יש לענות על כל השאלות בגיליון התשובות. מומלץ לא לחרוג מן המקום המוקצה.
- אם אינכם יודעים את התשובה, ניתן לכתוב 'לא יודע' ולקבל 20% מהניקוד על הסעיף/השאלה.

| | | |
|---|---|---|
| **שאלה 1:** | תחביר וסמנטיקה _____ | 35 נק |
| **שאלה 2:** | מערכת טיפוסים _____ | 20 נק |
| **שאלה 3:** | תכנות פונקציונאלי, CPS, רשימות עצלות | 35 נק |
| **שאלה 4:** | תכנות לוגי _____ | 20 נק |
| **סה"כ** | _____ | 110 נק |

**בהצלחה!**

---

## שאלה 1: תחביר וסמנטיקה [35 נקודות]

נתון כי לשפה L3 (ללא letrec) מומש אינטרפרטר על פי מודל הסביבות הרגיל (ללא הסביבה הרקורסיבית או ה-box), בנוסף לאינטרפרטר הקיים על פי מודל ההצבה.

נתונה הפונקציה map:

```
;; Signature: map(f,l)
;; Type: (T1->T2) * List(T1) -> List(T2)
;; Purpose: Apply f to all elements in l and return the list of the results.
(define map
  (lambda (f lst)
    (if (empty? lst)
        '()
        (cons (f (car lst))
              (map f (cdr lst))))))
```

א. תארו את תהליך החישוב במודל ההצבה/החלפה (substitution model) של הביטוי הבא, על ידי ציון הביטוי הנשלח כל פעם לחישוב (כלומר לפונקציה eval באינטרפרטר). <u>ציינו בתשובה רק את הביטויים מסוג if והפעלת הפונקציה map</u>

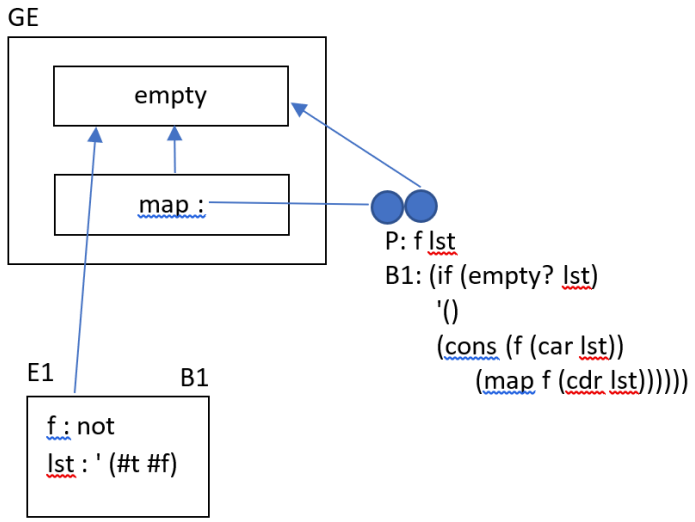
```
(map not '(#t #f))

Eval1: (map not '(#t #f))
Eval2: (if (empty? '(#t #f))
        '()
        (cons (not (car '(#t #f)))
              (map not (cdr '(#t #f)))))
Eval3: (map not (cdr '(#t #f)))
Eval4: (if (empty? '(#f))
        '()
        (cons (not (car '(#f)))
              (map not (cdr '(#f)))))
Eval5: (map not (cdr '(#f))
Eval6: (if (empty? '())
        '()
        (cons (not (car '()))
              (map not (cdr '()))))
Value: '(#f #t)
```

[5 נקודות]

ב. תארו את תהליך החישוב של הביטוי ב<u>מודל הסביבות</u> (environment model), על ידי שרטוט דיאגרמת הסביבות. יש לציין את המבנה הפנימי של הסביבה הגלובלית (את הפריימים שלה)

```
(map not '(#t #f))
```

[5 נקודות]




Error: Var map is not defined

ניקוד:
-1 הפעלה כמה פעמים של map (הרקורסיה לא נתמכת)
-1 אי ציון פריים ריק בסביבה
[שתי הנקודות הנ"ל באו תמיד ביחד תחת ההערה 'פריים ריק]
-1 שרשור של הסביבות הפעלות ה-map, ראינו בכיתה ש dynamic scoping הוא שגוי
[בהערות הבדיקה DS]

ג. כדי לתמוך בהפעלה של פונקציות רקורסיביות במודל הסביבות בשפה L3, הוחלט להוסיף את הצורה המיוחדת def-rec ולהגדיר סוג חדש של frame בסביבה - הממשק RecEnv.

<u>קוד הסביבה כפי שנלמד בכיתה</u>:

```
export type Env = EmptyEnv | ExtEnv | RecEnv;
export interface EmptyEnv {tag: "EmptyEnv" }
export interface ExtEnv {
    tag: "ExtEnv";
    vars: string[];
    vals: Value[];
    nextEnv: Env;
}
export interface RecEnv {
    tag: "RecEnv";
    vars: string[];
    paramss: VarDecl[][];
    bodiess: CExp[][];
    nextEnv: Env;
}
```

```
export const makeEmptyEnv = (): EmptyEnv => ({tag: "EmptyEnv"});
export const makeExtEnv = (vs: string[], vals: Value[], env: Env): ExtEnv =>
    {tag: "ExtEnv", vars: vs, vals: vals, nextEnv: env};
export const makeRecEnv = (vs: string[], paramss: VarDecl[][], bodiess:
CExp[][], env: Env): RecEnv =>
 {tag: "RecEnv", vars: vs, paramss: paramss, bodiess: bodiess, nextEnv: env};

const isEmptyEnv = (x: any): x is EmptyEnv => x.tag === "EmptyEnv";
const isExtEnv = (x: any): x is ExtEnv => x.tag === "ExtEnv";
const isRecEnv = (x: any): x is RecEnv => x.tag === "RecEnv";

export const isEnv = (x: any): x is Env => isEmptyEnv(x) || isExtEnv(x) ||
isRecEnv(x);

// Apply-env
export const applyEnv = (env: Env, v: string): Result<Value> =>
    isEmptyEnv(env) ? makeFailure(`var not found ${v}`) :
    isExtEnv(env) ? applyExtEnv(env, v) :
    applyRecEnv(env, v);

const applyExtEnv = (env: ExtEnv, v: string): Result<Value> =>
    env.vars.includes(v) ? makeOk(env.vals[env.vars.indexOf(v)]) :
    applyEnv(env.nextEnv, v);

const applyRecEnv = (env: RecEnv, v: string): Result<Value> =>
    env.vars.includes(v) ?
           makeOk(makeClosure(env.paramss[env.vars.indexOf(v)],
                              env.bodiess[env.vars.indexOf(v)],
                              env)) :
          applyEnv(env.nextEnv, v);
```

המבנה התחביري החדש DefRecExp:

```
export interface DefRecExp {tag: "DefRecExp"; var: VarDecl; val: ProcExp; }
export const makeDefRecExp = (v: VarDecl, val: ProcExp): DefRecExp =>
    {tag: "DefRecExp", var: v, val: val};
export const isDefRecExp = (x: any): x is DefRecExp => x.tag === "DefRecExp";
```

השלימו את קוד הפונקציה evalDefineExps:

```
const evalDefineExps = (def: Exp, exps: Exp[], env: Env): Result<Value> =>
 isDefineExp(def) ? bind(applicativeEval(def.val, env),
  (rhs: Value) => evalSequence(exps, makeExtEnv([def.var.var], [rhs], env))) :
 isDefRecExp(def) && (isProcExp(def.val)) ?
   evalExps(rest(exps),
        makeRecEnv([def.var.var],[def.val.args],[def.val.body],env));
 :
 makeFailure("Unexpected " + def);
```

[5 נקודות]

ד. האם ניתן לממש את הגישה המוצעת בסעיף הקודם (ג) עם אופרטור פרימיטיבי במקום צורה מיוחדת?
[3 נקודות]

לא, כי במקרה זה יחושב שם המשתנה (ה VarDecl) של מבנה ה def-rec כפרמטר של האופרטור הפרימיטיבי def-rec, ונקבל שגיאה (או שהוא לא מוגדר עדיין בסביבה, או שהא מוגדר ואז נקבל Value).

הניקוד ניתן ביחס לרמת הפירוט הנ"ל.

ה. סטודנטים בקורס הציעו לתמוך בהפעלות של פונקציות רקורסיביות במודל הסביבות מבלי להזדקק לפריים הרקורסיבי המיוחד, על ידי הוספה של פרמטר מטיפוס פונקציה לפונקציה הרקורסיבית.
השלימו את הקוד על פי הצעתם:

```
;; Signature: map(f,l)
;; Type: (T1->T2) * List(T1) * Type -> List(T2)
;; Purpose: Apply f to all elements in l and return the list of the results.
(define map
  (lambda (f lst rec)
    (if (empty? lst)
        '()
        (cons (f (car lst)) (rec f (cdr lst) rec)))))

(map not '(#t #f) map)
```

[5 נקודות]

ו. ציינו יתרון וחיסרון לגישה בסעיף ג ולגישה בסעיף ה לפתרון בעיית ההפעלה של פונקציות רקורסיביות במודל הסביבות.
[4 נקודות]

Y-combinator (סעיף ה): דורש עיצוב מיוחד של קוד המשתמשים, אם הם יכתבו את הקוד באופן הרגיל הוא לא יעבוד. מצד שני, אין צורך לבנות מחדש את הקלוז'ר בכל קריאה ל-map.
def-rec (סעיף ג): אין צורך בעיצוב מיוחד של קוד המשתמשים, אך נדרש ליצור קלוז'ר בכל הפעלה.

הניקוד ניתן ביחס לרמת הפירוט הנ"ל

ז. תארו בקצרה במילים את אופן מימוש הפונקציה defRec2Def, המקבלת מבנה תחבירי מסוג DefRecExp ומחזירה מבנה שקול מסוג DefExp (אם לא עניתם על סעיף ה, אתם יכולים להניח בתשובתכם שהוא ממומש)
[4 נקודות]

```
/*
Purpose: rewrite a single DefRecExp as a DefineExp form
Signature:  defRec2Def(defRecExp)
```

```
Type: [DefRecExp => DefineExp]
*/
const defRec2Def= (e: DefRecExp): DefineExp => {
   …
}
```

הביטוי המציין את ערך המשתנה במבנה ה def-rec הוא ProcExp, נמיר אותו ל-ProcExp אחר באופן הבא:

- הוספת פרמטר rec לרשימת הפרמטרים של ה ProcExp.
- הוספת הפרמטר rec לרשימת האופרדנים של כל הפעלה של הפונקציה (שמה ניתן על ידי שם המשתנה במבנה ה def-rec)

הערך המוחזר הוא ביטוי DefExp עם אותו שם משתנה אך עם ה-ProcExp החדש.

הקלתי בבדיקה של סעיף זה.

ח. דונו בקצרה בשאלה האם DefRecExp היא צורה מיוחדת
[4 נקודות]
אם מבצעים את ההמרה המתאורת בסעיף הקודם, אין צורך לטפל במבנה זה באינטרפרטר, כך שהיא אינה צורה מיוחדת.
אם לא מבצעים את ההמרה, יש לטפל במבנה החדש באינטרפרטר, כפי שנעשה בסעיף ג, ולכן זו צורה מיוחדת.
(בדומה לדיון על let בתרגיל)

ירדה נקודה למי שלא התייחס לשתי האפשרויות.

## שאלה 2: טיפוסים [20 נקודות, כל סעיף 5 נקודות]

א. התבוננו בביטוי הבא:

```
(define f
    (lambda ((x : number)) : number
        (if #f
            #t
            x)))
```

האם הביטוי הוא well typed (עובר type checking)? הסבירו

No for 2 reasons:

1. The IfExp returns two different type (boolean #t and number x).
2. The IfExp is the body of the procedure and it returns either a boolean or a number - which is not compatible with the declared return value of the function (number).

Note that the fact that the “then” branch of the if-exp is never reached (because the test is #f) does not change the conclusion - because:

1. type checking is a static analysis - it analyzes all sub-expressions of the program without analyzing which parts will be executed at runtime.
2. When type checking succeeds, we have proven that all possible executions will be type-safe - regardless of specific runtime parameters.

ב. התבוננו בביטוי הבא:

```
(define g
    (lambda ((x : number)) : number
        (if (= x 0)
            (throw “divide by zero”)
            (/ 1 x))))
```

האם הביטוי הוא well typed (עובר type checking)? הסבירו (throw הוא special form הזורק exception)

Yes - the expression is well-typed. In contrast to the example above (2א) because “throw” does not return a value - hence the type checker does not need to account for the type consistency of this branch. See the explanation in Lecture Notes: Type Checking | Principles of Programming Languages (bguppl.github.io):

**Soundness** The key property of a type system is a set of rules which determine whether a given expression in the language is type safe - that is, whether the evaluation of this expression will *never* lead to type errors.

Type safety is achieved by defining an analysis method called **type checking**. The goal of type checking is to verify that if an expression E is assigned type T, then, whenever E is computed, its value will be of type T. If the type system has this property, we say that it is **sound**.

Note that type checking does not guarantee that the program will always terminate (that would be a strong guarantee equivalent to solving the *Halting Problem*) **nor that it will not throw any exceptions**, such as divide by 0. It only guarantees that the program when it is evaluated to a proper value will not throw type errors and will return a value in the predicted type.

One way to think about how “throw” behaves with respect to typing is to compare it with Java: when a method in Java throws an Exception, we add this information to the signature - but it does NOT change the return type of the method.

ג. מה הסיבוכיות של ה-type checker של L5 שנלמד בכיתה (כפונקציה של N מספר הקודקודים ב-AST של הביטוי הנבדק)

It is O(N) since the type checker must analyze every node in the AST - and it analyzes every node only once. This is in contrast with the interpreter algorithm which can skip some nodes and traverse the same nodes multiple times.

ד. בשפה L51 שנחקרה בתרגיל 4 - record type מוגדר כ-component של user-defined disjoint union מוגדר כ-subtype של ה-user defined type.
למשל:

```
(define-type Shape
    (circle (radius : number))
    (rectangle (width : number) (height : number)))
```

ה-record בשם circle הוא subtype של ה-type בשם Shape.
התיחסו ל-type checking של ביטוי LetExp:

(let (( (<var>: <type>) <val>)) <body>)

מחשבים את type TVal של <val> — ואז בודקים התאמה בין TVal ו-<type>
האם בודקים:

- TVal is a subtype of <type> או:
- <type> is a subtype of TVal

הסבירו

We must check that TVal (the computed type of <val>) is a subtype of <type> (the declared type of the VarDecl for <var>. The constraint is that the value which is bound to <var> when evaluating the <body> of the LetExp must be certified to belong to its declared type <type>.
This is achieved when the computed type of <val> is a subset of the declared type <type>.
For example: (let (( (c : Shape) (make-circle 1) ) c) is well typed
(define f (lambda () : Shape (make-rectangle 1 1)))
(let (( (c : circle) (f) ) c) is not well typed.

## שאלה 3: תכנות פונקציונאלי, CPS, רשימות עצלות [35 נקודות]

**א. [15 נק']**

1. [3 נק] תנו שני מקרים בהם רצוי להשתמש ברשימות עצלות על פני רשימות רגילות, והסבירו מדוע.

עבודה עם רשימות אינסופיות - אין אפשרות לשמור את כל הרשימה בזיכרון.

קריאה של קובץ גדול יותר מהזיכרון הפנוי. כך לא נמלא את הזיכרון.

קריאה של נתונים מחיישן המשדר קריאות כל הזמן.

2. [3 נק] תנו שני מקרים בהם לא רצוי להשתמש ברשימות עצלות. הסבירו.

רשימות קצרות - נחסוך את זמן יצירת הרשימה.

איטרציה חוזרת של אותה הרשימה - כך ניצור אותה רק פעם אחת.

היזכרו ברשימה העצלה ones:

```
(define ones
  (cons-lzl 1
            (λ () ones)))
```

נשים לב כי הבנאי cons-lzl מקבל איבר, ואז ביטוי לאמבדה אשר אינו מקבל פרמטרים.

אנו רוצים להוסיף לתחביר של L5 בנאי עבור רשימות עצלות, כך שלא יקבל ביטוי לאמבדה ללא פרמטרים אלא את הזנב של הרשימה. למשל:

```
(define ones
  (cons-lzl 1 ones))
```

3. [3 נק']: האם אפשר לממש את cons-lzl באמצעות פרוצדורת משתמש? אם כן, ממשו אותה. אם לא, הסברו מדוע.

לא ניתן, כיוון שהפרמטרים יעברו הערכה לפני הפעלת הפרוצדורה, ואז תהיה לולאה אינסופית.

4. [3 נק'] האם אפשר לממש את cons-lzl באמצעות special form ? הסבירו (במידה וכן, אין צורך לממש, רק להסביר).

כן. חוק ההערכה המיוחד לא יפעיל את הפרוצדורה.

5. [3 נק'] האם אפשר לממש lexical abbreviation עבור cons-lzl? הסבירו (במידה וכן, אין צורך לממש, רק להסביר).

כן. הביטוי השקול יהיה ביטוי המכיל לאמבדה ללא פרמטרים.

**ב. [5 נק] גרסת CPS של הפונקציה take**

נתון המימוש של הפונקציה take:

```
(define take
  (lambda (lz-lst n)
    (if (or (= n 0) (empty-lzl? lz-lst))
        empty-lzl
        (cons (head lz-lst)
              (take (tail lz-lst) (- n 1))))))
```

המירו אותו לפונקציה בשיטת CPS (ניתן להתיחס ל-empty-lzl? כפרימיטיב)

```
(define take$
  (λ (lzl n cont)
    (if (or (= n 0) (empty-lzl? lzl))
        (cont empty-lzl)
        (take$
         (tail lzl)
         (- n 1)
         (λ (take-n-1)
           (cont (cons (head lzl)
                       take-n-1 )))))))
```

**ב. [6 נק'] רשימה עצלה של חזקות טבעיות**

השלימו את הקוד הבא עבור הפונקציה `exp-lzl` המקבלת מספר טבעי חיובי b ומחזירה רשימה עצלה אינסופית של החזקות הטבעיות: $b^1\ b^2\ b^3 \ldots$ .

לדוגמה:

```
(take (exp-lzl 2) 5) ⇒ '(2 4 8 16 32)

(define exp-lzl
  (λ (b)
    (letrec ([helper (λ (n)
                       (cons-lzl n
                                 (λ () (helper (* n b)))))])
      (helper b))))
```

**ג. [9 נק'] מיון מיזוג של רשימות עצלות**

השלימו את הקוד הבא של הפונקציה merge-lzl אשר מקבל שתי רשימות עצלות **ממוינות**, ומחזירה רשימה עצלה ממוינת של האיברים משתי הרשימות.

לדוגמה:

```
(take (merge-lzl (exp-lzl 2) (exp-lzl 3)) 10)
 ⇒ '(2 3 4 8 9 16 27 32 64 81)

(define merge-lzl
  (λ (lzl1 lzl2)
    (cond [(and (empty-lzl? lzl1) (empty-lzl? lzl2)) empty-lzl]
          [(empty-lzl? lzl1) lzl2]
          [(empty-lzl? lzl2) lzl1]
          [else
           (let ([h1 (head lzl1)]
                 [h2 (head lzl2)])
             (if (< h1 h2)
                 (cons-lzl h1 (λ () (merge-lzl (tail lzl1) lzl2)))
                 (cons-lzl h2 (λ () (merge-lzl lzl1 (tail lzl2))))))])))
```

## שאלה 4: תכנות לוגי [20 נקודות]

נתונים החוקים הלוגיים הבאים עבור ייצוג קבוצות כרשימות (כזכור, X \= Y מצליח כאשר X אינו unifiable עם Y):

```
member(X, [X|_]).
member(X, [_|Ys]) :- member(X, Ys).

not_member(_, []).   %r1
not_member(X, [Y|Ys]):- X \= Y, not_member(X,Ys).   %r2

% Signature: is_set(S)/1
% Purpose: check whether S is a set.
is_set([]).
is_set([X|Xs]):- not_member(X,Xs), is_set(Xs).

% Signature: intersection(S1,S2,S3)/3
% Purpose: S3 is the intersection (חיתוך) of S1 and S2.
%          The items in S3 are ordered according to their order in S1 and S2
intersection([],L,[]):- is_set(L).
intersection([X|Xs],L,[X|Ys]):-
   member(X,L),intersection(Xs,L,Ys),is_set([X|Xs]).
intersection([X|Xs],L,L2):-
   not_member(X,L),intersection(Xs,L,L2),is_set([X|Xs]).
```

א. הגדירו את היחסים הבאים <u>על בסיס היחס `intersection` בלבד</u>

```
% Signature: subset(S1,S2)/2
% Purpose: S1 is a subset (מוכל-שווה) of S2.
% ?- subset([1,2,3],[1,2,3,4])
%    true
% ?- subset([1,2,3],[1,2,3])
%    true
% ?- subset([1,2,3],[1,2])
%    false
subset(S1,S2):-intersection(S1,S2,S1).


% Signature: disjoint(S1,S2)/2
% Purpose: S1 and S2 are disjoint (זרות).
% ?-disjoint([1,2,3],[4,5,6])
%   true
% ?-disjoint([1,2,3],[3,5,6])
%   false
subset(S1,S2):-intersection(S1,S2,[]).
```

[6 נקודות]

ב. מדוע =\ היא צורה מיוחדת בשפה הלוגית?
[2 נקודות]

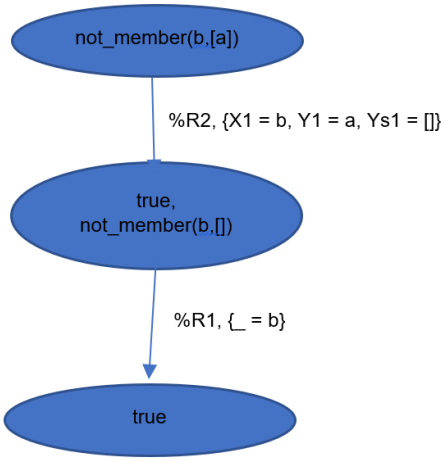
כי החישוב שלה אינו על פי ברירת המחדל של מציאת חוקים רלבנטיים על ידי הפעלת Rsel ופיתוח כל אחד מהם.

ג. ציירו את עצי ההוכחה עבור השאילתות הבאות:

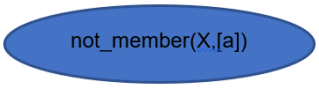
```
?-not_member(b,[b])
```


```
b \= b → false
```

```
?-not_member(b,[a])
```




```
?-not_member(X,[a])
```




```
X \= a → false
```

[8 נקודות]

ד. הסבירו בקצרה מדוע השאילתא האחרונה בסעיף ג אינה מניבה את התוצאה הלוגית הרצויה.
[2 נקודות]

כי היוניפיקציה X \= a מצליחה (מייצרת את ההצבה {X = a})

ה. הציעו דרך לפתור זאת על ידי הוספת צורה מיוחדת חדשה לשפה [2 נקודות]

נגדיר צורה מיוחדת חדשה =! המחזירה ערך אמת בכל מקרה למעט מקרים שבהם יש זהות בין שני ה terms משני הצדדים (a=a או X=X), כך ש- X \ = a יחזיר true, והשאילתא הנ"ל תצליח.