מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: ב
תאריך: 15/7/2021
שמות המרצים: מני אדלר, בן אייל, 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 _____	נק 30
שאלה 2:	מערכת טיפוסים _____	נק 25
שאלה 3:	מבני בקרה _____	נק 30
שאלה 4:	תכנות לוגי _____	נק 20
סה"כ	_____	נק 105

בהצלחה!

<u>שאלה 1</u>: תחביר וסמנטיקה [30 נקודות]

ראינו כי בחוק החישוב של if-exp, מחושב תת-הביטוי של ה-then **או** תת-הביטוי של ה-else בהתאם לערכו של ה-test, אך בכל מקרה **לא** מתבצע חישוב **גם** של ה-then **וגם** של ה-else.
בדומה לכך, ה-shortcut semantics עבור חישוב הפעלת אופרטורים - כמו and, or - נמנעת מלחשב בהכרח את כל הארגומנטים עבור הפעולה. לדוגמא:

- עבור הביטוי:

```
(and (> 2 3) (< 7 9))
```

מספיק לחשב את (3 2 >) מבלי לחשב את (9 7 <), כדי לקבוע כי ערך הביטוי כולו הוא #f.

- עבור הביטוי:

```
(or (< 7 9) (> 2 3))
```

מספיק לחשב את (9 7 <) מבלי לחשב את (3 2 >), כדי לקבוע כי ערך הביטוי כולו הוא #t.

פרוצדורת המשתמש my-and, מקבלת שני ביטויים ומחזירה #t אם הערך של כל אחד מהם הוא #t.

```
;; Signature: my-and(b1, b2)
;; Type: [Boolean * Boolean -> Boolean]
;; Purpose: return true if both b1 and b2 are true
;; Tests: (my-and (> 5 4) (> 2 1)) → true
;;        (my-and (> 4 5) (> 2 1)) → false
(define my-and
  (lambda (b1 b2)
     (if b1 b2 #f)))
```

א. האם ההפעלה:

```
(my-and (> 4 5) (> 2 1))
```

עומדת בקריטריון ה-shortcut semantics כאשר האינטרפרטר ממומש **ב-applicative order**? נמקו <u>בקצרה</u> (**במשפט אחד - תשובה ארוכה תיפסל**)
[5 נקודות]

ב. האם ההפעלה:

```
(my-and (> 4 5) (> 2 1))
```

עומדת בקריטריון ה-shortcut semantics כאשר האינטרפרטר ממומש **ב-normal order**? נמקו בקצרה (**במשפט אחד - תשובה ארוכה תיפסל**)
[5 נקודות]

ג. האם מימוש my-and **כאופרטור פרימיטיבי** בשפה יבטיח כי my-and תעמוד תמיד בקריטריון ה-shortcut semantics? אם כן, הסבירו כיצד; אם לא, נמקו בקצרה (**במשפט אחד - תשובה ארוכה תיפסל)**
[5 נקודות]

ד. האם מימוש my-and **כצורה מיוחדת** בשפה יבטיח כי my-and תעמוד תמיד בקריטריון ה-shortcut semantics? אם כן, הסבירו כיצד; אם לא, נמקו בקצרה (**במשפט אחד - תשובה ארוכה תיפסל)**
[5 נקודות]

ה. השלימו את קטעי הקוד הבאים, עבור הוספת הצורה המיוחדת my-and לשפה L3, על פי סמנטיקת ה shortcut:

תחביר (L3-ast.ts)

```
interface MyAndExp { tag : 'my-and';
                     ______________________________ }
const makeMyAnd = (__________________): MyAndExp =>
_________________________________________;
const isMyAnd = (x: any): _____________ => _______________________;
```

סמנטיקה (L3-eval.ts)

```
const applicativeEval = (exp: CExp, env: Env): Result<Value> =>
    isNumExp(exp) ? makeOk(exp.val) :
    isBoolExp(exp) ? makeOk(exp.val) :
    isStrExp(exp) ? makeOk(exp.val) :
    isPrimOp(exp) ? makeOk(exp) :
    isVarRef(exp) ? applyEnv(env, exp.var) :
    isLitExp(exp) ? makeOk(exp.val) :
    isMyAndExp(exp) ? evalMyAnd(exp, env) :
    isIfExp(exp) ? evalIf(exp, env) :
    isProcExp(exp) ? evalProc(exp, env) :
    isLetExp(exp) ? evalLet(exp, env) :
    isAppExp(exp) ? safe2(
     (proc: Value, args: Value[])=> applyProcedure(proc, args))
     (applicativeEval(exp.rator, env),
      mapResult(rand => applicativeEval(rand, env), exp.rands)):
    exp;
```

```
const evalMyAnd = (exp: MyAndExp, env: Env): Result<Value> =>
```

_____
_____
_____

[10 נקודות]

שאלה 2: טיפוסים [25 נקודות]

2.1 Typing Unifiers [4 נק]

חישבו את ה-MGU עבור ה-type expressions הבאים: (most general unifier)
אם לא קיים Unifier הסבירו למה.

2.1.1
TE1 = [T1 * [T1 -> T2] -> number]
TE2 = [[T3 -> T4] * [T5 -> number] -> number]

_____

_____

2.1.2
TE1 = [T1 * [T1 -> T2] -> number]
TE2 = [number * [symbol -> T3] -> number]

_____

_____

2.2 Type Inference [18 נקודות]

הגדרנו בתרגיל 4 את שפת L51, הכוללת מבנה class לפי ההגדרה הבאה:

```
<cexp> ::= ...
    | ( class : <TypeName>
      ( <varDecl>+ )
      ( <binding>+ ) ) / ClassExp(typeName:String,
                                  fields:VarDecl[], methods:Binding[]))
```

החישוב של ביטוי class מחזיר בנאי, המקבל פרמטרים עבור שדות המחלקה. הפעלה של בנאי זה עם פרמטרים מחזירה class value שטיפוסו מוגדר במבנה המחלקה (<TypeName>).
בהינתן class value (הערך המוחזר מבנאי המחלקה) c-value, וסמל m' המציין שם של מתודה, ההפעלה `(c-value 'm)` מחזירה את ה-closure של המתודה הרלבנטית.

```
(define pair
  (class : Tpair
    ((a : number)
     (b : number))
    ((first (lambda ()  a))
     (second (lambda () b))
     (scale (lambda (k) (pair (* k a) (* k b)))))))
(define (p34 : Tpair) (pair 3 4))
(define f (lambda ((x : Tpair)) (* ((x 'first)) ((x 'second)))))
(p34 'first) ; --> #<procedure>
((p34 'first)) ; --> 3
((p34 'scale) 2) ; --> #pair<6,8>
(f p34) ; --> 12
```

כדי לרשום את המשוואות בהמשך, היעזרו בהגדרת ה-typing rules שהוגדרו בתרגיל 4:

Typing rule define:

```
  For every: type environment _Tenv,
             variable declaration _x1
             expressions _e1 and
             type expressions _S1:
  If _Tenv o {_x1 : _S1} |- _e1 : _S1
  Then _Tenv |- (define _x1 _e1) : void
```

Typing rule Class Application:

```
  For every: type environment _Tenv,
             expressions _e1, _class_value
             symbols _m1, ..., _mk, k >= 0
             type expressions _U1, ..., _Uk

  // An expression (class_value 'method) returns a closure
  // whose type is defined in the class's type
  If _Tenv |- _class_value : ClassTExp[{_mi, _Ui}; i = 1..k],
     _Tenv |- _e1 : _mi
  Then _Tenv |- (_class_value _e1) : _Ui
```

Typing rule Class:

```
  For every: type environment _Tenv,
             variables _x1, ..., _xn, n >= 0
             symbols _m1, ..., _mk, k >= 0
             procedure expressions _p1, ..., _pk
             type variable _ct
             type expressions _S1, ...,_Sn, _U1, ...,_Uk
  // A class expression returns a class-value constructor
  // and the class type is bound to the ClassTExp
  If _Tenv o {_x1:_S1, ..., _xn:_Sn } |- _pi:_Ui for all i = 1..k,
  Then _Tenv |- (class : _ct ( _x1 ... _xn )
                              ((_m1 _p1) ... (_mk _pk))) :
                     [_S1 * ... * _Sn -> _ct]
                 _ct : ClassTExp[{_m1:_U1} ... {_mk : _Uk}]
```

כתבו את רשימת משתני טיפוס ואת רשימת המשוואות הנגזרות כאשר מבצעים את האלגוריתם של הסקת טיפוסים על הביטוים הבאים (אין צורך לפתור את המשוואות). עבור כל תת-ביטוי ברשימת משתני הטיפוס רשמו את ה-AST type של הביטוי לפי הגדרת ה-AST.

דוגמא עבור הביטוי:

```
(L5
  (define g (lambda (f x) (f x)))
  (g + 4))
```

Expression	Variable	Type
1. (L5 …)	T0	Program
2. (define g (lambda …))	T1	Define-Exp
3. (lambda (f x) (f x))	T2	Proc-Exp
4. (f x)	T3	App-Exp
5. f	Tf	VarRef
6. x	Tx	VarRef
7. (g + 4)	T4	App-Exp
8. g	Tg	VarRef
9. +	T+	PrimOp
10. 4	Tnum4	Num-Exp

Construct type equations:

Expression	Equation
1. (L5 …)	T0 = T4
2. (define g (lambda …))	T1 = void
	Tg = T2
3. (lambda (f x) (f x))	T2 = [Tf * Tx -> T3]
4. (f x)	Tf = [Tx -> T3]
5. (g + 4)	Tg = [T+ * Number -> T4]
6. +	T+ = [Number -> Number]
7. 4	Tnum4 = Number

ציינו את הטיפוסים והגדירו את המשוואות עבור התכנית הבאה:

```
(L51
  (define pair
    (class : Tpair
        ((a : number) (b : number))
        (scale (lambda (k) (pair (* k a) (* k b)))))))
  (((pair 3 4) 'scale) 2))
```

Expression	Variable	Type
1. (L51 …)	T0	Program

Expression	Equation
1. (L51 …)	_____

2.3 [3 נק]

כתבו את ה-type הנגזר עבור

```
((pair 3 4) 'scale)
```

בתוכנית:

_____

שאלה 3: מבני בקרה - ג'נרטורים, רשימות ועצים עצלים [30 נקודות]

א. כתבו generator בשם lazyReduce המקבל פונקציה של שני ארגומנטים reducer, איבר התחלתי init ומערך lst כך שבכל קריאה ל-next על ה-generator נקבל את ה-reduce המצטבר של הרשימה. לדוגמה:

```
const gen = lazyReduce((x: number, y: number) => x + y,
                       0, [1, 2, 3, 4, 5]);
console.log(gen.next()); // { value: 0, done: false }
console.log(gen.next()); // { value: 1, done: false }
console.log(gen.next()); // { value: 3, done: false }
console.log(gen.next()); // { value: 6, done: false }
console.log(gen.next()); // { value: 10, done: false }

function* lazyReduce<T1, T2>(reducer: (acc: T2, cur: T1) => T2,
                             init: T2,
                             lst: T1[]): Generator<T2> {
```

_____
_____
_____
_____
_____
_____
_____
_____

[10 נקודות]

ב. ממשו את הפרוצדורה append-lzl המקבלת שתי רשימות עצלות ומחזירה את הצירוף של שתיהן, בזו אחר זו, כרשימה עצלה [6 נקודות]

```
;; Lazy list ADT (constructor and accessors)
(define empty-lzl '())
(define cons-lzl cons)
(define head car)
(define tail
  (lambda (lzl)
    ((cdr lzl))))
(define empty-lzl? empty?)
```

```
;; Signature: lzl-append(lz1, lzl2)
;; Type: [Lzl(T) * Lzl(T) -> Lzl(T)]
(define lzl-append
  (lambda (lz1 lzl2)
      ____________________________________________________________
      ____________________________________________________________
      ____________________________________________________________
```

ג. בהרצאה ראינו את ה-ADT הבסיסי של עצים עצלים (אין צורך בפרוצדורות נוספות בשאלה זו):

```
(define make-lzt cons)
(define empty-lzt empty)
(define lzt->root car)
(define lzt->branches
  (lambda (lzt)
      ((cdr lzt))
    )
)
```

ג1. השלימו את הפרוצדורה lzt-filter המקבלת lazy tree ופרדיקט ומחזירה רשימה (רגילה) של כל הקודקודים בעץ המקיימים את הפרדיקט:

```
; Signature: lzt-filter(lzt, filterP)
; Type: [LZT(Node) * [Node -> Boolean] -> List(Node)]
; Purpose: Collect filtered nodes in a finite lazy tree
;          (depth-first order)
(define lzt-filter
  (lambda (lzt filterP)
    (letrec (
      (collect
        (lambda (lzt)
         (let ((children (flatmap collect (lzt->branches lzt))))
            (if ____________________
                _______________________
                _______________________)))))

      (if (empty-lzt? lzt)
          empty
          (collect lzt)))))
```

[6 נקודות]

ג2. השלימו את הפרוצדורה lzt-filter-lzl המקבלת lazy tree ופרדיקט ומחזירה רשימה עצלה של כל הקודקודים בעץ המקיימים את הפרדיקט:

```
; Signature: lzt-filter->lzl(lzt, filterP)
; Type: [LZT(Node) * [Node -> Boolean] -> Lzl(Node)]
; Purpose: Collect filtered nodes in depth-first-order and return the
results as a lazy list.
(define lzt-filter->lzl
  (lambda (lzt filterP)
    (letrec (

     (collect  ; [LZT(Node) -> LZL(Node)]
      (lambda (lzt)
        (if (filterP (lzt->root lzt))
            (make-lzl (____________________
                       ________________________________________
            (collect-in-trees (lzt->branches lzt)))))

     (collect-in-trees     ; [List(LZT(Node)) -> LZL(Node)]
       (lambda (lzts)
         (if (empty? lzts)
             empty-lzl
             (let ((first-lzl (collect (first lzts))))
               (if (empty-lzl? first-lzl)
                   (collect-in-trees (cdr lzts))
                   (append-lzl ________________________
                               ___________________________))))))))

      (if (empty-lzt? lzt)
          empty-lzl
          (collect lzt)))))
```

[8 נקודות]

<u>שאלה 4</u>: תכנות לוגי [20 נקודות]

א.

הפרוצדורה הראשית של האינטרפרטר לשפה הלוגית, שהוצג בכיתה, היא answer-query. כזכור, פרוצדורה זו מקבלת שאילתא ותוכנית ומחזירה את רשימת ההצבות עבורן השאילתא היא בעלת ערך אמת ביחס לתוכנית ('הפתרונות' לשאילתא).
במימוש הפרוצדורה נבנה עץ הוכחה על פי האלגוריתם, כאשר העץ מיוצג כעץ עצל (lazy tree): השורש של העץ מייצג את השאילתא הראשית, ופונקציית יצירת הבנים מייצרת את קודקוקי הבנים ע"פ האלגוריתם:

- בחירת goal מהשאילתא הנוכחית על ידי הפונקציה Gsel
- מציאת החוקים הרלבנטיים ל goal הנבחר, ואת ההצבות שעל פיהן נבחרו החוקים, על ידי הפונקציה Rsel
- בניית קודקודי הבנים, כל קודקוד ע"פ אחד החוקים וההצבה הנלווית אליו, עם שאילתא פשוטה יותר ע"פ חוק זה.

הסבירו <u>בקצרה</u> (**במשפט אחד - תשובה ארוכה תיפסל):**

א1. בהינתן עץ ההוכחה העצל, כיצד נעשה שימוש בפרוצדורה lzt-filter->lzl משאלה 3 בהמשך המימוש של answer-query?
א2. באילו מקרים ניתן להסתפק ב lzt-filter?

[10 נקודות]

ב.

נתונים הפרדיקטים הבאים עם דוגמאות הרצה (הפרמטר N הוא מספר צ'רץ'):

```
% Signature: take(List, N, Sublist)/3
% Purpose: Sublist is the first N elements from List

?- take([1, 2, 3, 4, 5], s(s(s(0))), X).
X = [1, 2, 3] ;
false.

?- take([1, 2], s(s(s(0))), X).
X = [1, 2] ;
false.

% Signature: pad(List, N, Padded)/3
% Purpose: Padded is List padded with *s to reach length N
```

```
?- pad([i, love, ppl], s(s(s(s(s(0))))), X).
X = [i, love, ppl, *, *] ;
false.

?- pad([i, love, ppl], s(0), X).
X = [i, love, ppl] ;
false.
```

ממשו את הפרדיקט ngrams/3 באמצעות הפרדיקטים הנ"ל. רשימה Words, מספר צ'רץ' N ורשימה של רשימות Ngrams עומדים ביחס אם Ngrams היא רשימת כל תתי-הרשימות העוקבות באורך N ב-Words. אם תת-רשימה לא מגיעה לאורך N, צריך להוסיף לה כוכביות (הסימבול *) כדי להגיע לאורך N. לדוגמה:

```
?- ngrams([i, love, ppl, very, much], s(s(s(0))), X).

X = [[i, love, ppl], [love, ppl, very], [ppl, very, much], [very,
much, *], [much, *, *]] ;
false.


% Signature: ngrams(Words, N, Ngrams)/3
% Purpose: Ngrams are padded N-grams of Words
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
```

[5 נקודות]

ג. בצעו יוניפיקציה על הביטויים הבאים. אם היוניפיקציה מצליחה, כתבו את ההצבה המתקבלת; אחרת, כתבו מדוע היוניפיקציה נכשלת. אין צורך לפרט את שלבי האלגוריתם.

```
unify( f([p(X), Y|Z], g([[V|V], V])),
       f([p(a), p(b), p(c), p(d)], g([[[]], []])) )
```

[5 נקודות]