מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: א
תאריך: 27/6/2023
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]

כדי לשחרר באופן יזום משתנים גלובליים שלא נדרשים יותר בתוכנית, הוצע להוסיף לשפה L4 (הממומשת עם box) צורה מיוחדת חדשה: undefine
undefine מקבל שם של משתנה, אם הוא מוגדר בסביבה הגלובלית הוא מוסר משם וחוזר true, אחרת חוזר false.
undefine הוא תמיד תת-ביטוי של program. לא ניתן להגדירו כתת-ביטוי של ביטויים אחרים.

```
(L4 (undefine x))
→ #f

(L4 (define x 7)
    (undefine x))
→ #t

(L4 (define x 7)
    x
    (undefine x)
    x)
→  { tag: 'Failure', message: 'Var not found: "x"' }
```

א. עדכנו את תחביר השפה L4 עם הצורה החדשה: התחביר הקונקרטי, התחביר המופשט, ומימוש המבנה התחבירי (ניתן לשנות הגדרות קיימות על ידי מחיקתן והגדרתן מחדש בשורות התשובה) [8 נקודות]

```
<program> ::= (L4 <exp>+) / Program(exps:List(exp))
<exp> ::= <define> | <cexp>  / DefExp | CExp
_____________________________________________________________
_____________________________________________________________
_____________________________________________________________
<define> ::= (define <var> <cexp>) / DefExp(var:VarDecl, val:CExp)
<var> ::= <identifier> / VarRef(var:string)
<cexp> ::=
  <number> / NumExp(val:number) |
  <boolean> / BoolExp(val:boolean) |
  <string> / StrExp(val:string) |
  (lambda (<var>*) <cexp>+) /
          ProcExp(args:VarDecl[], body:CExp[])) |
  (if <cexp> <cexp> <cexp>) /
          IfExp(test: CExp, then: CExp, alt: CExp) |
  (let (<binding>*) <cexp>+) /
          LetExp(bindings:Binding[], body:CExp[])) |
  (letrec (binding*) <cexp>+) /
```

```
          LetrecExp(bindings:Bindings[], body: CExp) |
  (quote <sexp> ) / LitExp(val:SExp) |
  (<cexp> <cexp>*) / AppExp(operator:CExp, operands:CExp[]))

<binding> ::= (<var> <cexp>) / Binding(var:VarDecl, val:Cexp)
<prim-op> ::= + | - | * | / | < | > | = | not | eq? | string=? |
              cons | car | cdr | list | pair? | list? |
              number? | boolean? | symbol? | string?
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<str-exp>  ::= "tokens*"
<var-ref>  ::= an identifier token
<var-decl> ::= an identifier token
<sexp>     ::= symbol | number | bool | string | ( <sexp>* )

export type UndefExp = {
   ________________
   ________________

}

export const makeUndefExp = (____________________________): UndefExp
=>
    ____________________________________________________;

export const isUndefExp = (________________) : ________________ =>
    ____________________________________;
```

ב. השלימו את מימוש הצורה החדשה באינטרפרטר.
[8 נקודות]

```
const evalUnDefExp = (undef: UndefExp): Result<boolean> =>
   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________
```

```
const evalDefineExp = (def: DefineExp): Result<undefined> =>
    bind(applicativeEval(def.val, theGlobalEnv), (rhs: Value) =>
        {
            globalEnvAddBinding(def.var.var, rhs);
            return makeOk(undefined);
        });

type GlobalEnv = {
    tag: "GlobalEnv";
    frame: Box<Frame>;
}

export type Frame = {
    tag: "Frame";
    fbindings: FBinding[];
}

export type FBinding = {
    tag: "FBinding";
    var: string;
    val: Box<Value>;
}

const globalEnvSetFrame = (ge: GlobalEnv, f: Frame): void =>
                                  setBox(ge.frame, f);

export const globalEnvAddBinding = (v: string, val: Value): void =>
    globalEnvSetFrame(theGlobalEnv,
            extendFrame(unbox(theGlobalEnv.frame), v, val));

const applyGlobalEnvBdg = (ge: GlobalEnv, v: string):
Result<FBinding> =>
    applyFrame(unbox(ge.frame), v);
```

ג. האם ניתן לממש את undefine כפרוצדורת משתמש במקום צורה מיוחדת? נמקו בקצרה [1 נקודה]

ד. האם ניתן לממש את undefine כאופרטור פרימטיבי? נמקו בקצרה [4 נקודות]

ה. הראו תרחיש שבו מבוצע '(undefine x)' בתוכנית, כאשר אחר כך אין שום התייחסות ל-x, ובכל זאת מתקבלת שגיאה (שלא היתה מתקבלת אם לא היינו מבצעים את undefine)
[5 נקודות]

```
(define x 7)
```

```
(undefine x)
```

ו. הציעו בקצרה דרך לפתור את הבעיה - יש לפרט (במילים) את השינויים שצריך לבצע באינטרפטר [6 נקודות]

ז. האם הבעיה שתארתם בסעיף ה תתרחש גם אם נממש את האינטרפרטר של L4 במודל ההצבה (עם box)? נמקו בקצרה.
[3 נקודות]

שאלה 2: טיפוסים [20 נקודות]

2.1 בסעיף זה נתייחס למערכת הטיפוסים שמוגדרת בתרגיל 4 - L5 עם תוספת של union. כזכור union מוגדר כ-compound-TExp לפי התחביר:

```
UnionTExp ::= (union <TExp> <TExp>) / union-te(components: list(te))
```

לדוגמה:

`(union number boolean)` defines the type which contains all the number values and the boolean values.

`(union number (union string boolean))` defines the type which contains all the number values, string values and boolean values.

א. השלימו את ה-L5 type annotations של הביטוי הבא [2 נקודות]

```
(define f
  (lambda ((x : (union number boolean))) : _____________________________
    (if (not (boolean? x))
      (if (> x 0)
          "Number"
          #t)
      #f))))
```

ב. האם הביטויים הבאים type-safe? נמקו בקצרה.
[3 נקודות]

```
(f (f #t))          ____________________________________________

(f (f "a"))         ____________________________________________

(f (if #t 1 #f))    ____________________________________________
```

2.2

נאמר כי type checker מקיים 'נָאוֹתוּת' (soundness) אם כאשר הוא קובע את הטיפוס של ביטוי נתון, קביעה זו תקפה לכל חישוב אפשרי.

א. האם ה-type checker של L5 מקיים נאותות?
[1 נקודה]

_____

_____

ב. ה-type checker אינו מקיים 'שלמוּת' (completeness), כי לעתים הוא מצביע על בעיית תאימות טיפוסים עבור ביטוי נתון, למרות שהביטוי עשוי להיות בטוח מבחינת הטיפוסים בזמן ריצה.

ציינו עבור שני המקרים הבאים האם יש בעיית חוסר שלמות ולמה:
[6 נקודות]

(f “a”) _____

_____

(f (f #t)) _____

_____

2.3 Type Inference with Equations

בצעו את השלבים הראשונים של אלגוריתם type inference with type equations על הביטוי הבא:

```
(lambda ([f : Tf] [x : Tx]) : T1
  (lambda ([g : Tg]) : T2
    (f (+ x (g #t)))))
```

א. השמה של TVar לכל קודקוד ב-AST [שתי נקודות]

Expression	Variable
(lambda (f x) (lambda (g) (f (+ x (g #t)))))	T0
(lambda (g) (f (+ x (g #t))))	T1

ב. רשימת המשוואות [6 נקודות]

Expression	Equation

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

3.1 מהי עמדת זנב (Tail Position)?
[2 נקודות]

3.2 מהו תהליך חישוב רקורסיבי?
[2 נקודות]

3.3 מהו תהליך חישוב איטרטיבי?
[2 נקודות]

3.4 הפונקציה הבאה מוצאת את המספר הגדול ביותר מתוך רשימה של מספרים:
[2 נקודות]

```
;; Signature: max(lst)
;; Purpose: Finds the largest element of a list of numbers.
;; If the given list is empty it produces an error.
;; Type: [List -> Number union Void]
;; Example: (max '(2 9 5)) ⇒ 9
(define max
  (lambda (lst)
    (cond
      ((empty? lst) (error "Empty list!"))
      ((empty? (cdr lst)) (car lst))
      (else (let ((max-cdr (max (cdr lst)))
                  (first (car lst)))
              (if (> first max-cdr)
```

```
                first
                max-cdr))))))
```

מהו התת-ביטוי בפונקציה הגורם לתהליך החישובי להיות רקורסיבי ולא איטרטיבי? נמקו.

_____

_____

_____

_____

3.5 השלימו את הקוד הבא, שהוא גירסה של הפונקציה max אשר מייצרת תהליך חישוב איטרטיבי:

[5 נקודות]

```
;; Signature: max-iter(lst)
;; Purpose: Finds the largest element of a number list.
;; If the given list is empty it produces an error.
;; The compuation is iterative.
;; Type: [List -> Number union Void]
;; Example: (max-iter '(2 9 5)) ⇒ 9
(define max-iter
  (lambda (lst)
    (letrec ((iter (lambda (lst max-so-far)
                     ________________________________________
                     ________________________________________
                     ________________________________________
                     ________________________________________
                     ________________________________________

      (if (empty? lst)
          (error "empty list")
          (iter (cdr lst) (car lst))))))
```

3.6 השלימו את הקוד הבא, שהוא גרסת CPS של max:

[5 נקודות]

```
;; Signature: max$(lst, succ-cont, fail-cont)
;; Purpose: Finds the largest element of a number list.
;; Type: [List * [Number -> T1] * [Empty -> Void] -> Number union
Void]
;; Example: (max$ '(2 9 5) id (lambda () (error "empty list"))) ⇒ 9
(define max$
  (lambda (lst succ-cont fail-cont)
    (cond ((empty? lst) (fail-cont))
```

```
((empty? (cdr lst)) (succ-cont (car lst)))
(else
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ____________________________________________________________
  ))))
```

3.7 כתבו ייתרון אחד שיש לגירסה האיטרטיבית על גרסת ה-CPS, והסבירו.

[3 נקודות]

_____
_____
_____
_____

3.8 כתבו יתרון אחד שיש לגרסת ה-CPS על פני הגרסה האיטרטיבית, והסבירו.

[3 נקודות]

_____
_____
_____
_____

3.9 כתבו שני יתרונות לעבודה עם **רשימות עצלות** על פני רשימות רגילות.

[4 נקודות]

_____
_____
_____
_____

3.10 השלימו את הקוד עבור הפונקציה הבאה המקבלת גבולות של תחום, a ו-b, ומיצרת **רשימת עצלה** של כל המספרים השלמים בתחום זה, לא כולל הגבול העליון של התחום.
השתמשו בממשק לעבודה עם רשימות עצלות:

[7 נקודות]

```
head, tail, empty-lzl?, cons-lzl
```

```
;; Signature: range-lzl(a,b)
;; Purpose: Creates a finite lazy list containing all
;;          the integers between a and b, not including b
;; Type: [Number * Number -> Lzl]
(define range-lzl
  (λ (a b)
    (letrec ((loop ____________________________________________
                   ____________________________________________
                   ____________________________________________
                   ____________________________________________
                   ____________________________________________
      (loop a))))
```

שאלה 4: תכנות לוגי [20 נקודות]

א. ממשו בשפה הלוגית את הפרוצדורה append2, הקובעת את היחס הבא: הפרמטר הראשון הוא רשימה של רשימות, והפרמטר השני הוא שרשור הרשימות האלה.
ניתן להניח שהפרמטר הראשון תמיד מוגדר בשאילתא.

```
% Signature: append2(Lists, List)/2
% Precondition: Lists is fully instantiated
%  (queries do not include variables in their first argument).

?- append2([ [a], [b,c] ], [a,b,c]).
true

?- append2([ [a], [b], [c] ], L).
L = [a,b,c]

?- append2([ [a], [[d,e]], [c] ], L).
L = [a, [d, e], c]
```

<u>אין</u> להגדיר חוקי עזר
<u>אין</u> להשתמש בפרוצדורות אחרות (בפרט לא ב-append ו-member שנלמדו בכיתה)
[10 נקודות]

ב. נתונה התוכנית:

```
element(a).    %r1
element(b).    %r2

swap(void,void).   %r3
swap(tree(Element, Left1, Right1), tree(Element,Left2, Right2))
 :- swap(Left1, Right2), swap(Right1, Left2).   %r4
```

ציירו את עץ ההוכחה עבור השאילתא הבאה:

```
?- element(X), element(Y), swap(tree(X,void,void),
tree(Y,void,void)).
```

[10 נקודות]

עץ ההוכחה: