מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: ב
תאריך: 19/7/2023
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]

הערה: בשאלה זו נתייחס בין היתר למימוש של מודל הסביבות עם normal-order.
במימוש זה:

- ה-bindings בפריימים של הפעלות הקלוז'רים קושרים שם משתנה לביטוי, במקום לערך כמו ב-applicative order שמומש בכיתה.
- הפונקציה applyEnv מחשבת את הביטוי המקושר לשם המשתנה הנתון ומחזירה את ערכו, במקום רק להחזיר את ערכו הנתון בפריים כמו ב-applicative order.
- הסביבה הגלובלית נשארת כשהייתה: ה-bindings קושרים שמות משתנים לערכים, ו-applyEnv מחזירה את הערך המקושר למשתנה נתון.

כזכור, הפונקציה and מקבלת שני פרמטרים ומחזירה true אם הערך של שניהם true, אחרת היא מחזירה false.
באינטרפרטר שכתבנו (סוג השפה, L1-L4, אינו רלבנטי לשאלה זו) מומשה הפונקציה and כאופרטור פרימיטיבי.

בסמנטיקת ה-shortcut של הפעולה and, אם ערכו של הפרמטר הראשון הוא false, הפרמטר השני אינו מחושב (כי אין בכך צורך) והפונקציה מחזירה מיד false.

א. עבור כל אחד מהאינטרפרטרים הבאים, ציינו האם סמנטיקת ה-shortcut מתקיימת בפועל ונמקו זאת בקצרה:

- מודל ההצבה/ההחלפה, applicative order (במימוש שנלמד בכיתה)

_____

- מודל ההצבה/ההחלפה, normal order (במימוש שנלמד בכיתה)

_____

- מודל הסביבות, applicative order (במימוש שנלמד בכיתה)

_____

- מודל הסביבות, normal order (במימוש המתואר בהערה למעלה)

_____

[8 נקודות]

ב. ממשו את and כפרוצדורת משתמש בשם my_and (בלי להשתמש באופרטורים פרימיטיביים, אין בכך צורך)

```
;; Type: boolean * booleab -> boolean
(define my-and
   (lambda (a b)

      _____________

     ______________

     ______________

  )
)
```

[3 נקודות]

ג. תארו את החישוב של התוכנית הבאה בכל אחד מהאינטרפרטרים הבאים, וציינו עם נימוק קצר האם מתקיימת סמנטיקת ה-shortcut עבורו. [16 נקודות]

בתיאור החישוב עבור מודל ההצבה/ההחלפה, ציינו את הביטוי הנשלח בכל שלב לפונקציית ה-eval (אין צורך לעשות זאת עבור ביטויי `define`).

לדוגמא:

עבור

```
(define square (lambda (x) (* x x))
(square 3)
```

יש לכתוב

```
eval: (square 3)
eval: square
eval: 3
eval: (* 3 3)
```

בתיאור החישוב עבור מודל הסביבות, ציירו את דיאגרמת הסביבות.

התוכנית:

```
(define x 1)
(define f
  (lambda (a) (> a x))

(my-and #f (f 2))
```

- מודל ההצבה/ההחלפה, applicative order (במימוש שנלמד בכיתה)

האם מתקיימת סמנטיקת ה-shortcut: _____

- מודל ההצבה/ההחלפה, normal order (במימוש שנלמד בכיתה)

האם מתקיימת סמנטיקת ה-shortcut: _____

- מודל הסביבות, applicative order (במימוש שנלמד בכיתה)

האם מתקיימת סמנטיקת ה-shortcut: _____

- מודל הסביבות, normal order (במימוש המתואר בהערה למעלה)

האם מתקיימת סמנטיקת ה-shortcut: _____

ד. כמסקנה מסעיפים א-ג, האם הייתם ממליצים לממש את and כצורה מיוחדת? התייחסו לשיקולים שונים.
[3 נקודות]

_____

_____

_____

ה. כדי להימנע קטגורית מבעיית מימוש סמנטיקת ה-shortcut בכל תצורה של האינטרפרטר, הציעה אחת הסטודנטיות לממש את my_and בסמנטיקת ה-shortcut כפרוצדורת משתמש תוך שימוש בחישוב עצל עבור הפרמטר השני.
ממשו מחדש את my-and, והדגימו כיצד יש להפעיל אותו על הביטוי מהתוכנית בסעיף ג:

```
;; Type: boolean * ___________________ -> boolean
(define my-and
   (lambda (a b)

     ____________

     _____________

     _____________

  )
)

~~(my-and #f (f 2))~~
(my-and #f _________________________)
```

[5 נקודות]

שאלה 2: טיפוסים [20 נקודות]

בשאלה זו נתייחס לשפה L52 שמבוסס על השפה L5 - שפה עם lambda, if, let, letrec, cons, car, cdr ועם type annotations לפי מערכת הטיפוסים הבאה:

```
<TExp> ::= <atomicTExp> | <compoundTExp> | <TVar>
<atomicTExp> ::= boolean | number | string | void
                 | any | never
<compoundTExp> ::= <procTExp>
                 | <typePredTExp>
                 | (union <TExp> <TExp>)
                 | (inter <TExp> <TExp>)
                 | (diff <TExp> <TExp>)
<procTExp> ::= ( <TExp>+ -> <TExp> ) | ( Empty -> <TExp> )
<typePredTExp> ::= ( <TExp> -> is <TExp> )
```

התוספות ב-L52 מעל L5 הן:

1. **any**: this type describes the set of all possible values (כל הערכים)
2. **never**: this type describes the empty set (קבוצה ריקה)
3. **(union <t1> <t2>)**: this type describes the set containing the union of the values in <t1> and <t2> (same as in HW4 in language L51). (איחוד)
4. **(inter <t1> <t2>)**: this type describes the set containing the intersection of the values in <t1> and <t2> (חיתוך)
5. **(diff <t1> <t2>)**: this type describes the set containing the values in <t1> that are not in <t2> (set difference) (הפרש בין קבוצות)
6. **(any -> is number)**: type predicates which are functions of one parameter that return a boolean value and inform the type checker that if the value is true, the parameter belongs to the type, else that the value does not belong to the type.

ה-type predicate מתנהג כמו type predicates ב-TypeScript. למשל:

```
(define (isNumber : (any -> is number))
  (lambda ((x : any)) : is number
    (number? x)))

Is semantically equivalent to the TypeScript function:

const isNumber = (x: any): x is number => typeof x === "number";
```

2.1 בנאי הטייפים union, inter, diff מוגדרים לפי חוקים של תורת הקבוצות. לכל ביטוי TExp הבא, רשמו ביטוי TExp יותר פשוט המכיל אותה קבוצה של ערכים (**רשמו בתחביר המדויק של Texp**)
[3 נק]

```
(inter number boolean) ____________________________________________

(inter never string)   ____________________________________________

(union never number)   ____________________________________________

(diff (union number string) string)   ___________________________

(inter number (union number boolean)) ___________________________

(inter (union boolean number) (union boolean string))  _________
```

2.2 השלימו את הקוד הבא כך שהוא יעבור type checking ב-L52 אך לא יעבור עם ה-type checker של L51. כזכור מערכת הטיפוסים של L51 מכילה union וזו של L52 מכילה בנוסף inter, diff, any, never ו-type predicates.
[3 נק]

```
;; Return type (any -> is number) defined in L52
;; In L51 the return type would be boolean
(define (isNumber : (any -> is number))
  (lambda ((x : any)) : is number
    (number? x)))

;; Function to complete
(define (good_in_L52 : ((union number boolean) -> ________________))

    (lambda ((x : (union number boolean))) : ___________________

        ____________________________________

        ____________________________________
    ))
```

2.3 עם ההגדרה של union ו-inter - הגדרנו את יחס הסדר החלקי isSubType בין ביטויי TExp.
הרחיבו את הפונקציה שהוגדרה ב-HW4 כדי לכסות את המקרים של any ו-never.
(אין צורך לטפל במקרים עם inter או diff)
[3 נק]

```
// Add cases for comparison with never and any
// isSubType(te1, te2) is true when te1 is included in te2
const isSubType = (te1: TExp, te2: TExp): boolean =>
  (isTVar(te1) && isTVar(te2)) ? equals(te1, te2) :
  isTVar(te1) ? true :
  isTVar(te2) ? true :
  // cases with never or any
```

```
  (isUnionTExp(te1) && isUnionTExp(te2)) ? isSubset(te1.components, te2.components) :
  isUnionTExp(te2) ? containsType(te2.components, te1) :
  (isProcTExp(te1) && isProcTExp(te2)) ? checkProcTExps(te1, te2) :
  isAtomicTExp(te1) ? equals(te1, te2) :
  false;
```

2.4 השלימו את ה-return type של פונקצית f ב-L52 - אם היא לא עוברת type checking רשמו Failure - אם היא כן עוברת type checking רשמו את הביטוי TExp של ה-return type בתחביר תקין של L52 (תחביר לא תקין לא יתקבל). נמקו.
[3 נק]

```
(define (is_number? : (any -> is number))
    (lambda ((x : any)) : is number
        (number? x)))

(define (is_boolean? : (any -> is boolean))
    (lambda ((x : any)) : is boolean
        (boolean? x)))
```

```
(define f
    (lambda ((x : (union number boolean))) : ________________________
        (if (is_number? x)
            (if (> x 0)
                "positive"
                "negative")
            (if (is_boolean? x)
                1
                x))))
```

נימוק:

_____

_____

_____

_____

_____

2.5 ה-typing rule של ProcExp ו-IfExp ב-L51 (השפה של HW4 שתומכת ב-union) הוא:
[6 נק]

```
// L51 ProcExp Typing rule:
For any expression Proc = (lambda ((x1 : t1) … (xn : tn)) : returnTE body)
and any type env tenv and any TExp t1, …, tn, returnTE

If   typeof(body, extend-tenv(x1=t1,...,xn=tn; tenv)) = returnTE
Then typeof<(lambda ((x1:t1) … (xn:tn)) : returnTE body), tenv> = [t1 * ... * tn -> returnTE]

// L51 IfExp Typing rule:
For any expression IfExp = (if test then else)
and any type env tenv and any TExp t1 and t2:

if typeof(test, tenv) = boolean
   typeof(then, tenv) = t1
   typeof(else, tenv) = t2
then typeof<(if test then else), tenv> = union(t1, t2)
```

הרחיבו את ה-typing rule ל-L52 עבור ביטויים מהסוג הבא:

```
(if (typePred var) then else)
```

כאשר typePred הוא ביטוי עם type שהוא type predicate ו-var הוא ביטוי מסוג varRef.

ניתן להשתמש ב-union, inter- ו-diff:

```
// L52 IfPredType Typing rule:
For any expression IfExp = (if test then else)
and any type env tenv and any TExp tv, t, t1 and t2:

if  test = (typePred var) and
    typeof(var, tenv) = tv and
    typeof(typePred, tenv) = TypePredTExp(tv -> is t) and
```

_____

_____

_____

2.6 איזה תכונה של ה-type checker מושפעת במידה ומתכנת כותב type predicate **שקרי**. למשל: **[2 נק]**

```
(define (is_number? : (any -> is number))
    (lambda ((x : any)) : is number
        (string? x)))    ;; - test string? for "is number"
```

הסבירו בקצרה עם דוגמה.

_____

_____

_____

_____

_____

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

3.1 [4 נק']
ציינו שני מאפיינים של פרדיגמת התכנות הפונקציונאלי.

3.2 [4 נק']
ציינו שני תרחישי שימוש (use cases) בהם עדיף שימוש בפרדיגמת **התכנות הפונקציונאלי**:

3.3 [4 נק']
ציינו שני תרחישי שימוש (use cases) בהם עדיף **להימנע** מפרדיגמת התכנות הפונקציונאלי.

3.4 [2 נק']
מהן מונאדות (monads) בתכנות פונקציונאלי?

3.5 [7 נק']

בסעיפים הבאים נבנה מספר פונקציות עזר על מנת לממש, בעזרת `reduce`, פונקציה אשר מקבלת רשימה וסופרת כמה פעמים מופיע כל איבר בה.

להלן חתימתה של `reduce`:

```
;; Signature: reduce(reducer, init, l)
;; Type: [(T1 * T2 -> T2) * T2 * List(T1) -> T2]
;; Purpose: Combine all the values of l using reducer
;;   (reduce + 0 '(1 2 3)) --> (+ 1 (+ 2 (+ 3 0)))
```

פונקציית העזר הראשונה אותה נממש בסעיף זה נקראת `dict-get`, והיא תמומש בסגנון CPS. היא מקבלת (1) רשימה, המשמשת על תקן מילון, כך שכל איבר ברשימה הוא זוג: מפתח וערך, (2) מפתח, (3) continuation עבור הצלחה ו-(4) continuation עבור כישלון. אם המפתח (2) נמצא בתוך המילון, אז חוזר הערך המשוייך למפתח. אם המפתח לא נמצא, אז מופעלת פונקציית הכישלון. מותר להניח כי מפתח מופיע רק פעם אחת במילון.

השלימו את קוד הפונקציה:

```
;; Signature: dict-get(dict key succ fail)
;; Type:
;; [List(Pair(T1,T2)) * T1 * [T2 -> T3] * [Empty -> T4] ->
;;         (union T3 T4)]
;; Purpose: Returns the value associated with the key.
;; Examples:
;;  (dict-get '((a . 1) (b . 2)) 'a id (λ () #f)) => 1
;;  (dict-get '((a . 1) (b . 2)) 'c id (λ () #f)) => #f
(define dict-get
  (λ (dict key succ fail)
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
```

3.6 [7 נק']

פונקציה נוספת שצריך לממש היא פונקציה שמעדכנת את המילון. היא מקבלת את המילון, מפתח וערך. אם המפתח קיים במילון, אזי הערך מתעדכן בערך החדש. אם המפתח לא קיים, אז המפתח והערך החדשים נכנסים למילון.

```
;; Signature: dict-set(dict key value)
;; Type: [List(Pair(T1,T2)) * T1 * T2 -> List(Pair(T1,T2))]
;; Purpose: sets a value for the given key.
;; (dict-set '() 'a 1) => '((a . 1))
;; (dict-set '((a . 1) (b . 2)) 'a 10) => '((a . 10) (b . 2))
(define dict-set
  (lambda (dict key value)
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
```

3.7 [7 נק']

עכשיו צריך לחבר את שתי הפונקציות יחדיו ולכתוב את פונקציית ה-reducer. הפונקציה מקבלת מילון ומפתח. אם המפתח קיים במילון יש להעלות את הערך ב-1. אם המפתח לא קיים, יש להוסיף מפתח חדש עם הערך 1. **רמז**: כאן זה בדיוק המקום להשתמש ב-cont ו-fail של dict-get:

```
;; Signature: value-counts(lst)
;; Type: [List(T1) -> List(Pair(T1,number))]
;; Purpose: Return counts of unique values.
;; (value-counts '(a b a a b c)) => '((c . 3) (b . 2) (a . 1)))
(define value-counts
  (λ (lst)
    (reduce
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
    ____________________________________________________________
      (list) lst)))
```

שאלה 4: תכנות לוגי [20 נקודות]

א. מצאו את ה-MGU של כל אחד מזוגות הביטויים הביטויים (אין צורך לתאר את הדרך). אם לא ניתן לבצע יוניפיקציה ביניהם, ציינו את הסיבה לכך.
[8 נקודות]

```
p([X|Xs],[a|Z],Xs)
p(L,L,[])
```

```
p([X|Xs],[a|Z],Xs)
p(L,L,L)
```

```
p(g([T]), g(T), g)
p(X, Y, T)
```

```
p(cons(a,cons(b,empty)))
p([a,b])
```

ב. כזכור, המספר ה-חי בסדרת פיבונאצ'י הוא סכום שני המספרים שלפניו. כאשר שני האיברים הראשונים בסדרה הם 1,1.

ב1. ממשו את הפרוצדורה fib/3, המגדירה את היחס הבא: שלושת המספרים הם שלושה מספרים עוקבים בסדרת פיבונאצ'י.
לייצוג מספרים נשתמש בפנקטור s (ה-Church numbers שנלמדו בכיתה)
[6 נקודות]

```
natural_number(zero).
natural_number(s(X)) :- natural_number(X).


plus(X, zero, X) :- natural_number(X).
plus(X, s(Y), s(Z)) :- plus(X, Y, Z).



%fib/3
_____________________________________________________

_____________________________________________________



?- fib(s(zero),s(zero),s(s(zero)))
true

?- fib(s(s(zero)),s(zero),N3)
false

?-fib(N1,N2,s(s(zero)))
N1 = s(zero), N2 = s(zero)
```

ב2. ממשו את הפרוצדורה fib/2, המגדירה את היחס הבא: הפרמטר הראשון הוא האינדקס של מספר בסדרת פיבונצאצ'י, והפרמטר השני הוא המספר הזה.
[6 נקודות]

```
%fib/2
_____________________________________________________

_____________________________________________________

_____________________________________________________


?- fib(zero, s(zero))   % fib0 = 1
true

?- fib(s(zero), s(zero))   % fib1 = 1
true

?- fib(s(s(s(s(zero)))), N)
N = s(s(s(s(s(s(s(s(zero))))))))    % fib4 = 8
```