מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: א
תאריך: 23/6/2021
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 _____	30 נק
שאלה 2:	מערכת טיפוסים _____	25 נק
שאלה 3:	מבני בקרה _____	30 נק
שאלה 4:	תכנות לוגי _____	20 נק
סה"כ	_____	105 נק

בהצלחה!

שאלה 1: תחביר וסמנטיקה [30 נקודות]

א. השלימו את הכתובות הלקסיקליות של המשתנים בקוד הבא: [6 נק']

```
(lambda (a b c)
 (if (eq? [b: __  __] [g: __  __])
     ((lambda (c)
        (cons [b: __  __] [c: __  __]))
      [a: __  __])
     [b: __  __]))
```

ב. הקוד הבא (שראינו בכיתה) מדגים במודל ההחלפה כיצד ללא renaming החישוב שגוי. בדוגמה זו יש שימוש במשתנה גלובלי z, להדגמת הבעיה:

```
(define z 1)

(((lambda (x)           ; 1
   (lambda (z) (x z))) ; 2
 (lambda (w) z))        ; 3
2)
```

תנו דוגמת קוד נוספת המדגימה את הבעיה, **שאינה משתמשת במשתנה גלובלי** [6 נק']

ג. הסבירו (ללא קוד) כיצד ניתן להשתמש בכתובות לקסיקליות כך שלא יהיה צורך ב-renaming **במודל ההחלפה**. הבחינו בין משתנים גלובליים ו-לא גלובליים בארגומנט שמוחלף ב-body של ה-closure (רמז: מה הבעיה שבגללה נדרש renaming, ואיך כתובות לקסיקליות פותרות אותה) [10 נק']

ד. הסבירו (ללא קוד) כיצד ניתן להשתמש בכתובות לקסיקליות **במודל הסביבות** על מנת לבצע גישה ישירה למשתנה בסביבה, ולא לחפש אותו בסביבות העוטפות [8 נק']

_____

_____

_____

_____

_____

_____

_____

שאלה 2: טיפוסים [25 נקודות]

2.1 Typing Statements [4 נק]

עבור ה-type statements הבאים, רשמו האם הם נכונים, ואם לא למה.

2.1.1 `{x : T1, y : T2, f : [T2 → T1]} |- (f y) : T1`

2.1.2 `{f : [T1 → T2], g : [T1 → T2], a : T1} |- (f (g a)) : T2`

2.2 Type Inference

2.2.1 [18 נקודות] כתבו את רשימת משתני טיפוס ורשימת המשוואות הנגזרות כאשר מריצים את האלגוריתם של הסקת טיפוסים על הביטוים הבאים (אין צורך לפתור את המשוואות).
עבור כל תת-ביטוי ברשימת משתני הטיפוס רשמו את ה-type של הביטוי לפי הגדרת ה-AST.

נתון ה-typing rule עבור ביטוי מסוג `define-exp`:

```
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
```

לדוגמא עבור הביטוי:

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

Expression	Variable	AST Type
`(L5 …)`	`T0`	`Program`
`(define g (lambda …))`	`T1`	`Define-Exp`
`(lambda (f x) (f x))`	`T2`	`Proc-Exp`
`(f x)`	`T3`	`App-Exp`
`f`	`Tf`	`VarRef`
`x`	`Tx`	`VarRef`
`(g + 4)`	`T4`	`App-Exp`
`g`	`Tg`	`VarRef`
`+`	`T+`	`PrimOp`
`4`	`Tnum4`	`Num-Exp`

Construct type equations:

Expression	Equation
`(L5 …)`	`T0 = T4`
`(define g (lambda …))`	`T1 = void`
	`Tg = T2`
`(lambda (f x) (f x))`	`T2 = [Tf * Tx -> T3]`
`(f x)`	`Tf = [Tx -> T3]`
`(g + 4)`	`Tg = [T+ * Tnum4 -> T4]`
`+`	`T+ = [Number -> Number]`
`4`	`Tnum4 = Number`

ציינו את הטיפוסים והגדירו את המשוואות עבור התכנית הבאה:

```
(L5
  (define f (lambda (n) (* n n)))
  (define g (lambda (h) (lambda (p) (h (h p)))))
  ((g f) 2))
```

Expression	Variable	AST Type
`(L5 …)`	`T0`	`Program`
`...`		

Expression	Equation
`(L5 …)`	`T0 = ...`
`...`	

2.2.2 [3 נק]

כתבו את ה-type הנגזר עבור g בתוכנית:

_____

שאלה 3: מבני בקרה [30 נקודות]

א. נתונה הפונקציה `add-at-end` אשר משרשרת איבר לסוף רשימה, ומחזירה את הרשימה החדשה:

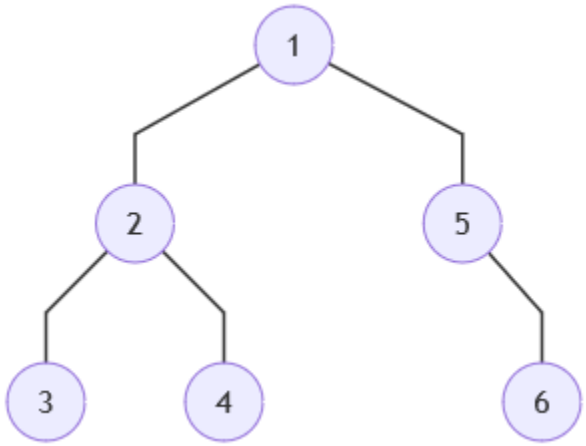
```
;; Signature: add-at-end(lst a)
;; Type: [ List(T1) * T1 -> List(T1) ]
;; Purpose: append element a at end of list lst
;; Tests: (add-at-end '(1 2 3) 4) => '(1 2 3 4)
(define add-at-end
 (lambda (lst a)
   (if (empty? lst) (list a)
       (cons (car lst)
             (add-at-end (cdr lst) a)))))
```

המירו את הפונקציה לגרסת CPS:
[5 נקודות]

```
;; Signature: add-at-end$(lst a c)
;; Type: [ List(T1) * T1 * [List(T1) -> T2] -> T2 ]
;; Purpose: append element a at end of list lst
;; Tests: (add-at-end$ '(1 2 3) 4 (lambda (x) x)) => '(1 2 3 4)
(define add-at-end$
 (lambda (lst a cont)
```

הוכיחו כי הפונ' `add-at-end` ו-`$add-at-end` שקולות CPS
[10 נקודות]

ב. כתבו בשפת Scheme את הפונקציה `$tree-reduce` המבצעת reduce על עץ בינארי בסדר pre-order. הפונקציה מקבלת $reducer (פונקציית CPS בינארית אשר מקבלת צובר acc ואיבר נוכחי cur), ערך התחלתי, עץ בינארי ו-continuation. בשאלה זו, עצים בינאריים ב-Scheme מיוצגים בצורת רשימה בה האיבר הראשון הוא שורש העץ, האיבר השני הוא תת-העץ השמאלי, והאיבר השלישי הוא תת-העץ הימני. לדוגמה, העץ הבינארי:



מיוצג ע"י הרשימה:

```
'(1 (2 (3 () ()) (4 () ())) (5 () (6 () ())))
```

דוגמאות הרצה על העץ הנ"ל:

```
> (define id (lambda (x) x))
> (tree-reduce$ (lambda (acc cur cont)
                (cont (string-append acc (number->string cur))))
                "" tree id)
"123456"

> (tree-reduce$ (lambda (acc cur cont) (cont (+ acc cur))) 0 tree id)
21
```

השתמשו בפונקציות הממשק הבאות לגישה לעץ:

```
(empty? tree)         ;; True if tree is '()
(tree->data tree)     ;; Returns the value of the node
(tree->left tree)     ;; Returns the left subtree of tree
(tree->right tree)    ;; Returns the right subtree of tree
```

[15 נקודות]

```
;; Signature: tree-reduce$(reducer$, init, tree, cont)
;; Type: ____________________________________________________________
(define tree-reduce$
  (lambda (reducer$ init tree cont)
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
```

שאלה 4: תכנות לוגי [20 נקודות]

א. ממשו את הפרדיקט take/3. רשימה List, מספר צ'רץ' N ותת-רשימה Sublist עומדים ביחס אם Sublist היא N האיברים הראשונים של List. אם יש פחות מ-N איברים ב-List, תת-הרשימה Sublist תהיה List כולה.

לדוגמה:

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

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

[5 נקודות]

ב. ממשו את הפרדיקט pad/3. רשימה List, מספר צ'רץ' N ורשימה Padded עומדים ביחס אם Padded היא הרשימה List בתוספת כוכביות (הביטוי *) לפי הצורך, כדי להגיע לאורך N. אם N קטן מאורך הרשימה, אין צורך בריפוד.

לדוגמה:

```
?- 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.

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

_____
_____
_____
_____
_____

[5 נקודות]

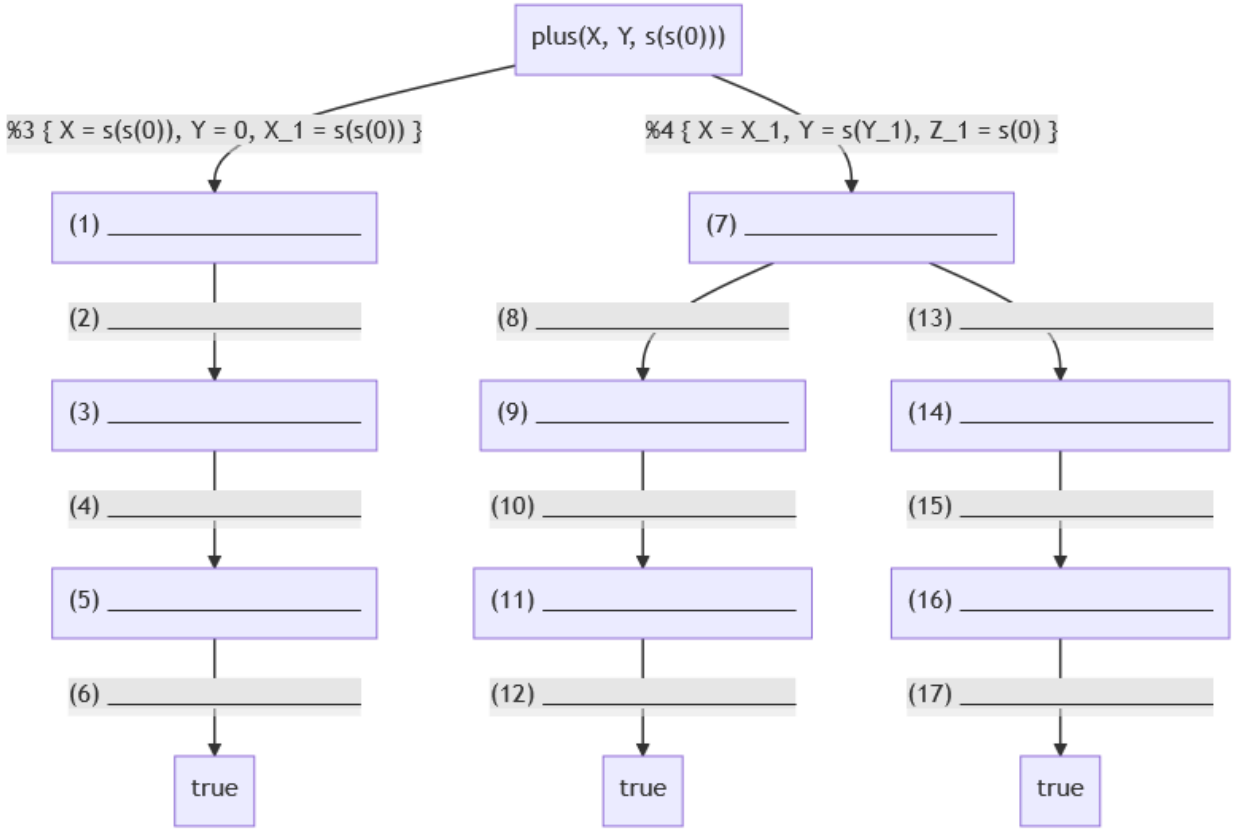
ג. בהינתן הפרוצדורות הבאות:

```
% Signature: natural_number(X)/1
% Purpose: X is a natural number in Church encoding
natural_number(0). %1
natural_number(s(X)) :- natural_number(X). %2

% Signature: plus(X, Y, Z)/3
% Purpose: Z = X + Y
plus(X, 0, X) :- natural_number(X). %3
plus(X, s(Y), s(Z)) :- plus(X, Y, Z). %4
```

השלימו את עץ ההוכחה הבא עבור השאילתה:

```
?- plus(X, Y, s(s(0))).
```



כתבו ליד כל מספר את ה-substitution/כלל המתאימים. שימו לב לכתוב ליד כל substitution את מספר הכלל הרלוונטי; כפי שמופיע בחלקים המלאים בראש העץ. [10 נקודות]