עבודה 5 בעקרונות שפות תכנות

# שאלה 1

## שאלה 1.1

### סעיף a

שני lazy lists (נסמנם lzl1, lzl2) שקולים זה לזה אם:

1. שתי הרשימות ריקות (הן '() )

או;

* 1. במקרה ששתי הרשימות העצלות אינסופיות:  
     לכל n, הפעלה n פעמים של הפונקציה tail על lzl1 ולאחר מכן הפעלת head על התוצאה, והפעלה n פעמים של הפונקציה tail על lzl2 ולאחר מכן הפעלת head על התוצאה, תניב את אותו הערך. כלומר,

(head (tail (tail (… (tail lzl1))))) = (head (tail (tail (… (tail lzl2)))))

* 1. במקרה ששתי הרשימות העצלות אינן אינסופיות:  
     קיים n כך שהפעלה n פעמים של הפונקציה tail על lzl1 תניב את הערך '(), וגם הפעלה n פעמים של הפונקציה tail על lzl2 תניב את הערך '(). כלומר,

(tail (tail (tail (… (tail lzl1))))) = (tail (tail (tail (… (tail lzl2))))) = '()

וגם לכל i<n, מתקיים התנאי שבסעיף a.

### סעיף b

נראה כי even-squares-1 שקול ל even-squares-2 על פי ההגדרה מסעיף a:

באופן כללי, הרשימה העצלה even-squares-1:

* מייצרת מספר טבעי.
* מעלה אותו בריבוע.
* בודקת האם הוא זוגי, ואם כן מחזירה אותו; אחרת – מייצרת את המספר הטבעי הבא.

והרשימה העצלה even-squares-2:

* מייצרת מספר טבעי.
* בודקת האם הוא זוגי, ואם לא, מייצרת את המספר הטבעי הבא; אחרת -
* מעלה אותו בריבוע ומחזירה אותו.

אבחנה: מספר טבעי m הוא זוגי אם ורק אם הריבוע שלו m2 הוא זוגי (כיוון שמכפלת שני מספרים זוגיים היא זוגית, ומכפלת שני מספרים אי זוגיים היא אי זוגית), ולכן כל אחת מהרשימות מחזירות את אותם הערכים (והן אינסופיות).

כעת נראה שהן מחזירות את הערכים באותו הסדר, בהתאם להגדרה בסעיף a, בעזרת אינדוקציה.

בסיס n=0:

שתי הרשימות מייצרות את המספרים הטבעיים באותו האופן (ע"י שימוש ב integers-from) החל מ-0.

לכן הפעלת tail 0 פעמים ואחריה הפעלת head על הערך המוחזר, היא למעשה:

(head even-squares-1)

(head even-squares-2)

נתבונן בהפעלה (head even-squares-1):

* תחילה even-squares-1 מפעילה את (integers-from 0) וחוזר הערך 0.
* לאחר מכן היא מעלה אותו בריבוע בעזרת הפעלת lzl-map עם (lambda (x) (\* x x) כאשר x=0.
* לבסוף היא מפעילה את lzl-filter על הערך המוחזר (שהוא שווה ל- 0) ובודקת האם הוא זוגי.
* מאחר שהוא זוגי, היא מחזירה אותו.

נתבונן בהפעלה (head even-squares-2):

* תחילה even-squares-2 מפעילה את (integers-from 0) וחוזר הערך 0.
* לאחר מכאן היא מפעילה את lzl-filter על הערך המוחזר (שהוא שווה ל- 0) ובודקת האם הוא זוגי.
* מאחר שהוא זוגי, היא מעבירה אותו לפונקציה lzl-map, שמפעילה את הפונקציה (lambda (x) (\* x x) עם x=0.
* לבסוף, חוזר הערך 0.

שתי הרשימות מחזירות את המספר 0, ומקרה הבסיס מתקיים.

הנחת האינדוקציה:

עבור כל אחת מהרשימות: הפעלת הפונקציה tail, n-1 פעמים, על הרשימה ולאחר מכן הפעלת head על הערך המוחזר, מניבה את אותו הערך.

צעד האינדוקציה:

נתבונן בהפעלה הn-ית של (head (tail …(tail even-squares-1)…) – הפעלת tail n פעמים על הרשימה ולאחר מכן הפעלה של head:

* תחילה even-squares-1 מפעילה את lzl-map על הערך המוחזר מ

(integers-from 0).

* בסוף הריצה הקודמת של even-squares-1 חזר ערך מסוים m2, לכן בריצה הקודמת של even-squares-1 ההפעלה האחרונה של (integers-from 0) החזירה את הערך m.
* לכן כעת הערך שהוחזר מ (integers-from 0) היה m+1.
* כאמור, על m+1 מופעלת lzl-map שמעלה אותו בריבוע בעזרת (lambda (x) (\* x x) כאשר x=m+1.
* מאחר ש m היה זוגי, אזי m+1 הוא אי זוגי, ולכן גם (m+1)2 אי זוגי.
* לכן, הפרדיקט (lambda (x) (= (modulo x 2) 0) שהוא הפרמטר לפונקציה lzl-filter מחזיר false, ולכן lzl-filter מפעילה את tail פעם נוספת על even-squares-1.
* כעת הערך שהוחזר מ (integers-from 0) הוא m+2.
* ובסופו של דבר הערך שיחזור הוא (m+2)2.

נתבונן בהפעלה הn-ית של (head (tail …(tail even-squares-2)…) – הפעלת tail n פעמים על הרשימה ולאחר מכן הפעלה של head:

* מהנחת האינדוקציה, הערך שהוחזר על ידי even-squares-2 בהפעלה הn-1 זהה לערך שחזר על ידי even-squares-1 בהפעלה הn-1 ולכן הוא שווה ל m2.
* לכן כעת הערך שהוחזר מ (integers-from 0) היה m+1.
* מאחר ש m היה זוגי, אזי m+1 הוא אי זוגי.
* לכן, הפרדיקט (lambda (x) (= (modulo x 2) 0) שהוא הפרמטר לפונקציה lzl-filter מחזיר false, ולכן lzl-filter מפעילה את tail פעם נוספת על even-squares-2.
* כעת הערך שהוחזר מ (integers-from 0) הוא m+2.
* על m+2 מופעלת lzl-map שמעלה אותו בריבוע בעזרת (lambda (x) (\* x x) כאשר x=m+2.
* ובסופו של דבר הערך שיחזור הוא (m+2)2.

לפיכך, שתי הרשימות מחזירות את אותו הערך לאחר הפעלת tail מספר זהה של פעמים (ואחריו head) על כל אחת מהן.

לכן, התנאי שהגדרנו בסעיף a מתקיים, ומכך שהרשימות שקולות זו לזו.

# שאלה 2

### סעיף a

נגדיר פרוצדורה f מטיפוס [T1 \* … \* Tn -> (U1 union U2)] ופרוצדורה f$ מטיפוס

[T1 \* … \* Tn \* [U1 -> U1] \* [Empty -> U2] -> (U1 union U2)]

נאמר ש-f$ היא גרסת ה- Success-Fail-Continuationsהשקולה ל-f, אם:

ל-f ול-f$ יש אותה תמונה (range), ולכל succ, fail מתקיים אחד מהבאים:

* אם הפעלת (f x1 x2 … xn) מחזירה ערך מטיפוס U1 (התבצע flow של ריצה תקינה) אז הפעלת (succ (f x1 x2 … xn)) והפעלת (f$ x1 x2 … xn succ fail) מחזירות את אותו הערך (מטיפוס U1).
* אם הפעלת (f x1 x2 … xn) מגיעה ל-flow שמייצג שגיאה ומחזירה ערך מטיפוס U2, אז גם הפעלת (fail) מחזירה את אותו הערך (מטיפוס U2) וגם הפעלת  
  (f$ x1 x2 … xn succ fail) מחזירה את אותו הערך (מטיפוס U2).
* אם הפעלת (f x1 x2 … xn) לא מסתיימת אז גם הפעלת  
  (f$ x1 x2 … xn succ fail) לא מסתיימת.

הבהרה: התייחסנו למקרה בו flow של שגיאה מיוצג ע"י החזרת ערך מטיפוס U2, בדומה ניתן להגדיר מקרה בו שגיאה מיוצגת ע"י זריקת error, ואז ב-flow של כישלון הפעלת  
(f x1 x2 … xn), הפעלת (fail) והפעלת (f$ x1 x2 … xn succ fail) תזרוק את אותה השגיאה.

### סעיף d

להשלים

נראה כי get-value ו-get-value$ שקולות עפ"י ההגדרה בסעיף a.

כיוון שהפרוצדורה get-value היא רקורסיבית, נוכיח באינדוקציה על אורך הרשימה  
assoc-list, נסמנו n.

בסיס האינדוקציה: n=0. אז בהכרח assoc-list = '().

יהיו key סימבול כלשהו ו-success, fail פרוצדורות כלשהן. אז:

הערך key לא נמצא ב-assoc-list לכן החישוב מסתיים בשגיאה. כלומר:

a-e [ (get-value assoc-list key) ] 🡺\* 'fail

ואכן מתקיים

a-e [ (get-value$ assoc-list key success fail) ] 🡺\* a-e [ (fail) ]

מכאן לפי ההגדרה בסעיף a, הפרוצדורות שקולות עבור n=0.

הנחת האינדוקציה: עבור הטענה מתקיימת לכל . כלומר לכל רשימה assoc-list מאורך i מתקיים שלכל key, succ, fail הפרוצדורות get-value ו-get-value$ שקולות.

צעד האינדוקציה: יהא , אזי:

מקרה א': הפעלת (get-value assoc-list key) מסתיימת בהצלחה ומחזירה ערך שנסמנו val.

להשלים

a-e [ (get-value assoc-list key) ] 🡺\* val

מהנחת האינדוקציה, נקבל:

a-e [ ((lambda (res) (c (\* n res))) (fact (- n 1))) ] ==>\*

a-e [ (c (\* n (fact (- n 1)))) ] =

a-e [ (c (fact n)) ]

(define get-value

  (lambda (assoc-list key)

    (if (empty? assoc-list)

      'fail

      (if (eq? (car (car assoc-list)) key)

        (cdr (car assoc-list))

        (get-value (cdr assoc-list) key)

      )

    )

  )

)

(define get-value$

  (lambda (assoc-list key success fail)

   (if (empty? assoc-list)

      (fail)

      (if (eq? (car (car assoc-list)) key)

        (success (cdr (car assoc-list)))

        (get-value$ (cdr assoc-list) key success fail)

      )

    )

  )

)

# שאלה 3

## שאלה 3.1

### סעיף a

תוצאת ה-unification היא כישלון, נפרט.

נבצע את שלבי אלגוריתם Unify על:

Unify [ t ( s ( s ) , G , H , p, t ( E ) , s ) ,

t ( s ( H ) , G , p , p, t ( E ) , K ) ]

ראשית נגדיר substitution ריק: {}.

שני ה-composite terms הם atomic formula עם הפרדיקט t, שמכיל terms שונים. כמות ה-terms בכל אחד מה-atomic formulas שווה, לכן נשווה בין כל ה-terms שבתוך ה-t-ים.

ניצור את המשוואות: s(s) = s(H), G = G, H = p, p = p, t(E) = t(E), s = K.

נתבונן במשוואה הראשונה. נבצע unification בין s(s) ל- s(H). גם כאן מדובר בהשוואה של שני atomic formulas, ה-predicate שווה וכמות ה-terms ב-atomic formula שווה, אז נשווה בין ה-terms שבתוך ה-s-ים. נקבל H=s. נוסיף אותו ל-substitution שלנו ונקבל: {H=s}.

נשארנו עם המשוואות: G = G, H = p, p = p, t(E) = t(E), s = K.

נפעיל את ה-substitution שלנו על G=G ונוסיף את התוצאה ל-substitution שלנו, ונקבל:  
{H=s, G=G}.

נשארנו עם המשוואות: H = p, p = p, t(E) = t(E), s = K.

נפעיל את ה-substitution שלנו על H=p ונקבל כישלון, שכן H=s וגם H=p.

### סעיף b

נבצע את שלבי אלגוריתם Unify על:

Unify [ g ( c , v ( U ) , g , G , U , E , v ( M ) ) ,

g ( c , M , g ,v ( M ) , v ( G ) , g , v ( M ) ]

ראשית נגדיר substitution ריק: {}.

שני ה-composite terms הם atomic formula עם הפרדיקט g, שמכיל terms שונים. כמות ה-terms בכל אחד מה-atomic formulas שווה, לכן נשווה בין כל ה-terms שבתוך ה-g-ים.

ניצור את המשוואות:  
c = c, v(U) = M, g = g, G = v(M), U = v(G), E = g, v(M) = v(M)

### סעיף c

תוצאת ה-unification היא כישלון, נפרט.

נבצע את שלבי אלגוריתם Unify על:

Unify s ( [ v | [ [ v | V ] | A ] ] ) ,

s ( [ v | [ v | A ] ] )

ראשית נגדיר substitution ריק: {}.

שני ה-composite terms הם atomic formula עם הפרדיקט s, שמכיל terms שונים. כמות ה-terms בכל אחד מה-atomic formulas שווה, לכן נשווה בין כל ה-terms שבתוך ה-s-ים.

ניצור את המשוואה: [v | [ [v | V] | A ]]= [ v | [v | A]].

נפעיל את ה-substitution הריק שלנו על המשוואה, הterms יישארו ללא שינוי.

אנו מבצעים unification לשני זוגות, ולכן כעת נבצע unification בין כל זוג איברים מאותם האינדקסים. נקבל את המשוואות:

v=v, [[v|V] | A] = [v | A]

הפעלת ה-substitution שלנו על המשוואה הראשונה לא משנה את הterms. ונקבל כעת את ה-substitution:

{v=v}

כעת נעבור למשוואה השנייה.

הפעלת ה-substitution שלנו לא משנה את הterms.

גם כאן, אנחנו מבצעים unification לשני זוגות, ולכן נבצע unification בין כל זוג איברים מאותם האינדקסים. נקבל את המשוואות:

v=[v|V], A=A

נתבונן במשוואה הראשונה.

הפעלת ה-substitution שלנו לא משנה את הterms.

אך כעת נוצר מעגל שכן אותו הsymbol (v) מופיע בשני צידי המשוואה, ולכן הunification נכשל.