## שאלה 3

### A

נראה סדרת חישוב לפי call by value (כאשרa,b הם value)

test (and tru fls) a b =

(λl. λm. λn. l m n) ( (λb. λc. b c (λt. λf. f)) (λt. λf. t) (λt. λf. f)) a b =>

(λl. λm. λn. l m n) ( (λc. (λt. λf. t) c (λt. λf. f)) (λt. λf. f)) a b =>

(λl. λm. λn. l m n) ((λt. λf. t) (λt. λf. f)) (λt. λf. f)) a b =>

(λl. λm. λn. l m n) ((λf. (λt. λf. f)) (λt. λf. f)) a b =>

(λl. λm. λn. l m n) ( λt. λf. f) a b =>

(λm. λn. ( λt. λf. f) m n) a b =>

(λn. ( λt. λf. f) a n) b =>

( λt. λf. f) a b =>

(λf. f) b =>

b

### B

**OR**

λb. λc. b tru c

אם b הוא tru נקבל:

λc. (λt. λf. t) tru c => (λt. λf. t) tru c => (λf. tru) c => tru

לכן נחזיר tru, לא משנה מה c. (כך גם OR עובד)

אם b הוא fls נקבל:

λc (λt. λf. f) tru c => (λt. λf. f) tru c => (λf. f) c => c

נחזיר את c.  
זה נכון, כי כעת אם c הוא tru נקבל tru, ואם c הוא als נקבל fls – בדיוק כמו שOR היה מחזיר.

**NOT**

λb. b fls tru

אם b הוא tru נקבל: fls [(λt. λf. t) fls tru => (λf. fls) tru => fls ]

אם b הוא fls נקבל: tru [(λt. λf. f) fls tru => (λf. f) tru => tru]

**XOR**

λb. λc. not (and (b c tru) (c b tru))

נראה שכאשר הערכים של b ו-c שונים נקבל tru, וכאשר עם זהים נקבל fls:  
(הערה: מיכוון שפעולת הפקודות not, and ,tru ,fls כבר ברורה לנו, [מטרת הסעיף היא לא בניית סדרת חישוב] הריצה אליהן בחישובים למטה היא מידית מבלי לרשום את החישוב המלא צעד אחר צעד. לכן הסימן החץ מופיע עם כוכבית)

אם b,c הם tru נקבל:

λb. λc. not (and (b c tru) (c b tru)) =>\* not (and (tru c tru) (tru b tru)) =>\*

not (and c b) =>\* not tru =>\*

fls

אם b,c הם fls נקבל:

λb. λc. not (and (b c tru) (c b tru)) =>\* not (and (fls c tru) (fls b tru)) =>\*

not (and tru tru) =>\* not tru =>\*

fls

אם b הוא tru וc הוא fls, נקבל:

λb. λc. not (and (b c tru) (c b tru)) =>\* not (and (tru c tru) (fls b tru)) =>\*

not (and c tru) =>\* not fls =>\*

tru

אם b הוא fls וc הוא tru, נקבל:

λb. λc. not (and (b c tru) (c b tru)) =>\* not (and (fls c tru) (tru b tru)) =>\*

not (and tru b) =>\* not fls =>\*

tru

## שאלה 4

### A

כן, התוצאה שווה ל-c1.

scc = λn. λs. λz. s (n s z)

c0 = λs. λz. z

**scc c0 =** (λn. λs. λz. s (n s z)) λs. λz. z => (E-AppAbs)

λs. λz. s ((λs. λz. z) s z) => (E-Abs)

λs. λz. s z = c1

### B

לא, התוצאה לא שווה ל-c1

(λn. λs. λz. s (n s z)) λs. λz. z => (E-AppAbs)

λs. λz. s ((λs. λz. z) s z) ⇏

התוצאה שקולה ל-c1 במובן שהפעלה של הפונקציה הנ"ל על ערכים x,y כלשהם, תוביל לתוצאה זהה להפעלתם על הפונקציה c1:

λs. λz. s ((λs. λz. z) s z) x y => λz. x ((λs. λz. z) x z) y=>

x ((λs. λz. z) x y( => x ((λz. z) y) => x y .

c1 = (λs. λz. s z) x y => (λz. x z) y => x y

קיבלנו אותה תוצאה לכל x,y , לכן הפונקציות שקולות.

### C

**newScc = λn. λs. λz. c1 s (n s z)**

כאשר הערך של n הינו Church number – אותו נקדם באחד.

השתמשנו בהגדרה של plus ( = λm. λn. λs. λz. m s (n s z))- אשר מקבל שני מספרי church ומחבר אותם, וקבענו בו את m (המחובר הראשון) להיות c1, כלומר נחבר את מספר הקלט (n) תמיד עם c1 – קיבלנו scc.

### D

times = λm. λn. m (plus n) c0

**power = λm. λn. n (times m) c1**

כאשר הפרמטר הראשון (m) הוא המעריך, והפרמטר השני (n) הוא החזקה.

בנינו זאת בצורה דומה לבניית פונקצית הכפל: פעולת חזקה היא הכפלה של המעריך בעצמו כמספר הפעמים של החזקה.

לדוגמא, נבחר מעריך c2 וחזקה c3, נקבל:

(λm. λn. n (times m) c1) c2 c3 =>

(λn. n (times c2) c1) c3 =>

c3 (times c2) c1 =

(λs. λz. s s s z) (times c2) c1 =>

(λz. (times c2) (times c2) (times c2) z) c1 =>

ניתן לראות שאכן קיבלנו שהפרמטר m מייצג את מספר ההכפלות של הפרמטר n בעצמו, כאשר ההכפלה הראשונה היא ב-c1 (כלומר הכפלה שלא משנה את התוצאה) – נמשיך בדוגמא:

(times c2) (times c2) (times c2) c1 =>

(times c2) (times c2) c2 =>

(times c2) c4 =>

c8

### E

**isZero = λm. m (λb. fls) tru**

הגדרה: הפונקציה isZero מקבל מספר church (שהוא בפועל פונקציה שמקבלת 2 פרמטרים) ומעבירה לו כפרמטר ראשון פונקציה שמקבלת פרמטר 1 ומחזירה **(λb. fls)** false, וכפרמטר שני את הערך tru.  
  
הסבר: נבחר מספר church כלשהו ck.  
המקרה היחידי בו אין שימוש בפרמטר הראשון **(λb. fls)**במספר ck, הוא כאשר ck=c0.   
במקרה הזה פשוט חוזר הערך השני: לכן בהפעלת פונקציה isZero על c0 נקבל tru.  
בכל מקרה אחר (ck>c0), נעשה גם שימוש בפרמטר הראשון שהעברנו, ולכן תמיד נקבל fls. (נריץ את הפונקציה **λb. fls** כמספר הפעמים של המספר ck – כל פעם על תוצאת הפונקציה בריצה הקודמת [התוצאה היא תמיד fls]– ולכן התוצאה הסופית היא fls).