עקרונות שפות תכנות:

תרגיל 3:

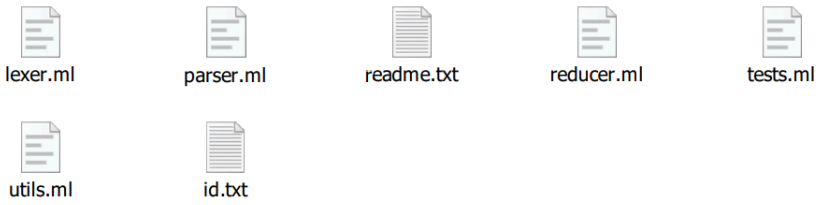
תאריך הגשה: 25.2.24

הוראות הגשה: ההגשה בזוגות / כיחידים דרך מערכת הסאבמיט. כל זוג נדרש לחשוב, לפתור ולכתוב את התרגיל בעצמו. יש לקרוא הוראות אלא בקפידה. הגשה שלא על פי הוראות אלה תוביל להורדת ניקוד.

מה להגיש:

בו יש את הפתרון לחלק א - ex3.pdf

את כל הקבצים שקיבלתם בחלק ב עם התיקונים שביצעתם (הגישו גם קבצים שלא שיניתם בכלל):



קובץ עם השם משתמש בסבמיט ות.ז של כל אחד מהמגישים באופן הבא:

id.txt:

```
301111111 NOFRI
209111111 YOSIM
```

כל הקבצים צריכים להיות בקובץ zip אחד בשם: ex4.zip

חלק א': תחשיב למדא:

שאלה 1:

חשבו את הביטויים (בצעו רדוקציית בטא) הבאים עד כמה שניתן (אן יש גזירה אינסופית הסבירו במילים למה יש גזירה אין סופית)

א. $(\lambda z.\, z)\ (\lambda y.\, y\ y)\ (\lambda x.\, x\ a)$

ב. $(((\lambda x.\, \lambda y.\, (x\ y))\ (\lambda y.\, y))w)$

ג. $(\lambda x.\, y)\ ((\lambda y.\ \ y\ y\ y)(\lambda x.\ \ x\ x\ x))$

ד. $rec\ fact\ 2$ (לפי התרגול, אפשר בדרך שלי או של צביקה 🙂)

שאלה 2:

א. כתבו חישוב call by value:

$$test\ (or\ tru\ fls)\ a\ b$$

כאשר a,b הם ערכים כלשהם.

ב. כתבו ביטוי בתחשיב למדא עבור האופרטור: nor (חשבו מה מקבל ומה מבצע)

ג. חשבו בעזרת הביטוי את (לא לדלג על שום שלב של רדוקיית הבטא):

$$nor\ tru\ fls$$

$$nor\ tru\ tru$$

שאלה 3:

א. חשבו את $succ\ c_1$ בעזרת Call-By-Name האם התוצאה היא c2?

ב. חשבו את $succ\ c_1$ בעזרת Call-By-Value האם התוצאה היא c1?

ג. הגדירו את פונקציית iseven, מקבלת מספר ומחזירה tru אם המספר הוא זוגי ו-fls אם הוא אי זוגי.

ד. חשבו את: (השתמשו ברדוקציית בטא, לפי כל סדר שתבחרו)

$$iseven\ 3$$

$$iseven\ 4$$

<u>שאלה 4:</u>

בכל אחת מהקביעות הבאות, קיבעו מהו הטיפוס T כך שהקביעה מתקיימת. הוכיחו זאת תוך שימוש בכללי הגזירה.

א. $f{:}Bool \to Bool \vdash (f\ (\text{if true then false else true})){:}T$

ב. $f{:}Bool \to Bool \vdash (\lambda x{:}\,Bool.\ \ f\ (\text{if x then false else true})){:}T$

ג. $\vdash (\lambda x{:}\,Bool.\ \ \lambda y{:}\,T.\ \ y\ x){:}Bool \to T \to Bool \to Bool$

בס"ד

- **<u>חלק ב: מפרש לתחשיב למדא</u>**

בתרגיל זה נבנה Parser ו-Interpeter לתחשיב למדא בשפת Ocaml.

בתרגיל נעשה שימוש בOcaml. כל הפתרונות צריכים ולהתקמפל ללא שגיאות עם הפקודות שמפורטות ב-readme.txt.

כל השינויים בקבצים צריכים להיות במקומות המסומנים בהם. אין לשנות בקבצים דבר מלבד במקומות אלה. יש לבדוק את הקוד שכתבת על דוגמאות נוספות ולוודא את נכונותו.

מומלץ לקרוא את התרגיל עד סופו לפני שמתחילים לפתור אותו.

שימו לב שאופן הרצת התרגיל מוסבר בקובץ readme.txt המצורף.

בשאלה זו נבנה Parser לתחשיב למדא מורחב שכולל גם let expression. התחביר הקונקרטי מוגדר ע"י הדקדוק הבא. אנו משתמשים בסמל \ במקום למדא (λ):

```
t ::= id | (\id. t) | (t1 t2) | (t) | let id = t1 in t2
```

שימו לב שהדקדוק מחייב סוגריים מסביב לפעולות abstraction ו-application.
לדוגמא, המחרוזת הבאה היא מילה חוקית בשפה:

```
let tru = (\t. (\f. t)) in
let fls = (\t. (\f. f)) in
let and = (\b. (\b. ((b c) fls))) in
((and tru) fls)
```

הייצוג הפנימי לביטויים בשפה הוא AST, שניתן ע"י התחביר האבסטרקטי הבא:

```
Term ::= id | \id. term | term1 term2
```

הקובץ lexer.ml מכיל את ה-Lexer המלא עבור שפה זו (אין לשנות קובץ זה), ומגדיר את הטיפוס token.
הקובץ parser.ml מגדיר את הטיפוס הבא, שמשמש לייצוג ה-AST (אין לשנות טיפוס זה):

```
Type term = Variable of string
          | Abstraction of string * term
          | Application of term * term
```

בקובץ parser.ml ממומשים הפונקציות הבאות:

```
parse_term : token list -> term * token list
parse : string -> term
format_term : term -> string
```

- הפונקציה parse_term מקבלת רשימה של tokens ומחזירה term ורשימה של tokens שנשארו. הפונקציה זורקת SyntaxError במידה וה-parsing נכשל.

הפונקציה מטפלת בביטויים מהצורה let x = t1 in t2 ע"י ייצוגם בתחביר האבסטרקטי כך:

(\x. t2) t1

(השתכנעו שזהו אכן ייצוג שמשמר את המשמעות של let expressions כפי שאנו מכירים אותם).

- הפונקציה parse מקבלת מחרוזת ומחזירה את ה-term שהיא מייצגת, או זורקת SyntaxError במידה והמחרוזת אינה מכילה מילה בשפה (לפי הדקדוק של התחביר הקונקרטי).

- הפונקציה format_term מקבלת term ומחזירה ייצוג שלו באמצעות מחרוזת (לדוגמה לצורך הדפסה). הייצוג הינו מילה חוקית בשפה – כך שהפעולה של parse על התוצאה של format_term מחזירה term זהה ל-term המקורי.

בקובץ reducer.ml נבנה interpreter עבור תחשיב למדא בשלבים, בשאלות הבאות.

בקובץ זה נשתמש במודול StringSet מהקובץ utils.ml כדי לייצג קבוצות של מחרוזות. המודול מכיל פונקציות עבור פעולות נפוצות על קבוצות (איחוד, הוספת איבר, הוצאת איבר, וכו'), והתיעוד שלו זמין ב: OCaml library : Set.S. הקובץ utils.ml גם מכיל פונקציה נוחה להדפסת קבוצות של מחרוזות.

א. הוסיפו לקובץ reducer.ml את הפונקציה:

```
fv : term -> StringSet.t
```

שמקבלת term ומחזירה את קבוצת המשתנים החופשיים בו. את הקבוצה יש לייצג באמצעות המודול StringSet (שמגיע מ-utils.ml). כזכור, את קבוצת המשתנים החופשיים ניתן להגדיר באופן אינדוקטיבי כך:

FV(x) = {x}

FV($\lambda x.\ t$) = FV(t) – {x}

FV($t_1\ t_2$) = FV(t_1) ∪ FV(t_2)

ב. לצורך מימוש alpha-conversion, אנו זקוקים לפונקציה שתחזיר שם של משתנה חדש שאינו בקבוצה של משתנים בשימוש. לצורך כך הקובץ reducer.ml מכיל את הערך possible_variables: string list, שמכיל רשימה של שמות משתנים אפשריים.
בקובץ reducer.ml הוספנו את הפונקציה:

```
fresh_var : StringSet.t -> string
```

הפונקציה מקבלת קבוצה של שמות משתנים בשימוש, ומחזירה שם חדש מתוך הרשימה possible_variables. במידה וכל השמות ברשימה בשימוש, הפונקציה זורקת OutOfVariablesError.

<u>הוסיפו לקובץ reducer.ml את הפונקציה:</u>

```
substitute : string -> term -> term -> term
```

על פונקציה זו לממש החלפה (substitution) כולל alpha-conversion במקרה הצורך. סדר הפרמטרים הוא כזה שהביטוי substitute "x" t1 t2 יחזיר את:

t2 [x -> t1]

כלומר t2 כאשר כל המופעים של המשתנה x הוחלפו ב-t1 (ולא להיפך!)

הפונקציה צריכה לבצע את ההחלפה בכל מקרה, תוך שהיא מבצעת alpha-conversion במקרה הצורך. את פעולת הפונקציה ניתן להגדיר אינדוקטיבית באופן הבא:

ההגדרה של Substitution היא באופן הבא:

$x[x \mapsto s]$	$= s$	
$y[x \mapsto s]$	$= y$	if $y \neq x$
$(\lambda x.\ t_1)[x \mapsto s]$	$= \lambda x.t_1$	
$(\lambda y.\ t_1)[x \mapsto s]$	$= \lambda y.t_1[x \mapsto s]$	if $y \neq x$ and $y \notin FV(s)$
$(\lambda y.\ t_1)[x \mapsto s]$	$= \lambda z.(t_1[y \mapsto z])\ [x \mapsto s]$	if $y \neq x$ and $y \in FV(s)$
$(t_1\ t_2)[x \mapsto s]$	$= t_1[x \mapsto s]\ t_2[x \mapsto s]$	

ג. הוסיפו לקובץ reducer.ml את הפונקציה:

reduce_cbv : term -> term option

על פונקציה זו לממש צעד אחד של חישוב (reduction) לפי סמנטיקת call-by-value. הפונקציה מחזירה ערך מטיפוס term option, כיוון שלא על כל term ניתן לבצע reduction. משמעות ערך החזרה היא:

(reduce_cbv t) = Some t' if t->t' in call-by-value

(reduce_cbv t) = None if t is not reducible in call-by-value

הפונקציה צריכה לממש את הכללים שנלמדו בשיעור ובתרגול, כאשר הערכים היחידים הם abstractions.

ד. הוסיפו לקובץ reducer.ml את הפונקציה:

reduce_cbn : term -> term option

על פונקציה זו לממש צעד אחד של חישוב (reduction) לפי סמנטיקת call-by-name. הפונקציה מחזירה ערך מטיפוס term option, ומשמעות ערך החזרה הוא בדיוק כמו ההסבר בסעיף הקודם. הפונקציה צריכה לממש את הכללים שנלמדו בשיעור ובתרגול, כאשר הערכים היחידים הם abstractions.

ה. הקובץ tests.ml מכיל את הפונקציה הבאה:

evaluate : verbose:bool -> (term -> term option) -> term -> term

פונקציה זו מקבלת את אחת הפונקציות שמימשתם בסעיפים ה-ו. בנוסף היא מקבלת term, ומחשבת אותו, איטרטיבית עד לצורה שהיא irreducible תוך שימוש בפונקציה הנתונה. אם הפרמטר verbose הוא true, הפונקציה גם מדפיסה את תהליך החישוב. הקובץ tests.ml גם מכיל קלטי בדיקה ראשוניים והרצות בדיקה בסמנטיקות השונות.

הרחיבו את הקובץ כדי שיכלול בדיקות נוספות, והשתמשו בו במהלך הפיתוח של כל השאלות הקודמות כדי לבדוק את המימוש.