בס"ד

שפות תכנות – תרגיל 3

תאריך הגשה: 22.01.2025

הוראות הגשה: ההגשה בזוגות או לבד. כל זוג נדרש לחשוב, לפתור ולכתוב את התרגיל בעצמו. מותר להתייעץ עם סטודנטים אחרים אך חל איסור מוחלט להחזיק ולהיעזר בתרגיל כתוב של זוג אחר. יש לקרוא הוראות אלא בקפידה, הגשה שלא על פי הוראות אלה תוביל להורדת ניקוד ולא יתקבלו על כך ערעורים!

חומר עזר מומלץ: כדאי להבין היטב את הרצאות ותרגולים הקשורים לתחשיב למדא. קישור לתרגולים (המצגות נמצאות בתיאור הסרטון):

https://www.youtube.com/watch?v=EMJmw_oAAec&list=PLaMkJ2Pfx92I7DbMteYLYmMDyn3N0dDIT&index=5

מה להגיש:

ex3.pdf - בו יש את הפתרון לחלק א

את כל הקבצים של חלק ב עם התיקונים שביצעתם (הגישו גם קבצים שלא נעשה בהם שינוי):

tests.ml reducer.ml readme.txt parser.ml lexer.ml

id.txt utils.ml

בקובץ id.txt יש לכתוב את הת.ז והשמות של המגישים.

יש להגיש את כל הקבצים בקובץ zip בשם ex3.zip.

- <u>חלק א: חישובים בתחשיב למדא</u>

בכל חישוב למדא לביטוי אין לדלג על אף צעד של רדוקציית בטא.

<u>שאלה 1:</u> חישוב ביטויי למדא

חשבו את הביטויים הבאים עד כמה שניתן (אם יש גזירה אינסופית הסבירו במילים למה):

<u>תזכורת:</u> כאשר אין סוגריים, הביטוי x y z = (x y) z.

1. $(\lambda z. z)\ \ (\lambda y.\ \ y\ \ y)\ (\lambda x. x\ a)$
2. $(\lambda x.\ \ \lambda y. x\ y\ y)\ (\lambda a. a)\ b$
3. $(((\lambda x. \lambda y. (x\ y))\ (\lambda y. y))\ w)$

גיזרו את הביטוי בסעיף 4 פעם אחת בעזרת call-by-value ופעם שנייה בעזרת call-by-name. האם קיבלתם את אותה תוצאה?

4. $(\lambda x. y)\ ((\lambda y. y\ y\ y)\ (\lambda \mathrm{x.\ x\ x\ x}))$

<u>שאלה 2:</u> תחשיב למדא לביטויים בוליאניים

נתונות ההגדרות הבאות (שחלקן ראינו בתרגול):

tru = $\lambda t. \lambda f.\ \ t$

fls = $\lambda t. \lambda f.\ \ f$

test = $\lambda l. \lambda m. \lambda n.\ \ l\ m\ n$

or = $\lambda b. \lambda c.\ \ b\ tru\ c$

1. כתבו חישוב עם אסטרטגייתcall-by-value   לביטוי:

   test (or tru fls) a b

   כאשר a, b הם ערכים כלשהם.

2. כתבו ביטוי בתחשיב למדא עבור nand.
3. חשבו בעזרת הביטוי את (לא לדלג על שום שלב של רדוקציית הבטא):

   nand tru fls

   nand tru tru

שאלה 3: תחשיב למדא לביטויים אריתמטיים

בתרגול ראינו את ההגדרות הבאות למספרים טבעיים ופעולות אריתמטיות:

$c_0 = \lambda s. \lambda z.\ \ z$

$c_1 = \lambda s. \lambda z.\ \ s\ z$

$c_2 = \lambda s. \lambda z.\ \ s\ (s\ z)$

$c_3 = \lambda s. \lambda z.\ \ s\ (s\ (s\ z))$

...

$succ = \lambda n.\ \ \lambda s.\ \ \lambda z.\ \ s\ (n\ s\ z)$

$plus = \lambda m.\ \ \lambda n.\ \ \lambda s.\ \ \lambda z.\ \ m\ s\ (n\ s\ z)$

$times = \lambda m. \lambda n.\ \ m\ (plus\ n)\ c_0$

iszero = $\lambda m.\ \ m\ \ (\lambda x. fls)\ tru$

1. חשבו את succ c_0 בעזרת Call-By-Name, האם התוצאה היא c_1 ?
2. חשבו את $c_0$succ בעזרת Call-By-Value, האם התוצאה היא c_1 ?
3. הגדירות פונקציית isodd, שתקבל מספר טבעי כפי שקודדנו בהגדרת השאלה ותחזיר tru אם המספר הוא אי-זוגי ו-fls אם המספר זוגי.
4. חשבו ע"י רדוקציית בטא את:

isodd 4

isodd 5

שאלה 4: Simply Typed Lambda Calculus

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

1. f:Bool → Bool ⊢ (f (if true then false else true)):T
2. f:Bool → Bool ⊢ ($\lambda x: Bool.$ f (if x then false else true)):T
3. ⊢ ($\lambda x: Bool.\ \ \lambda y: T.$ y x):Bool→ T →Bool→Bool

שאלה 5: **Simply Types Lambda Calculus**

השלימו את החסר בהוכחה הבאה.

בס"ד

למת ההתקדמות

אם t מקיים $\vdash t:T$ עבור איזשהו T, והוא ללא משתנים חופשיים אז או שהוא ערך או שיש s עם $t \to s$.

הוכחה

באינדוקציה על הגזירה של $\vdash t:T$. נפריד למקרים לפי הכלל האחרון שמופיע בגזירה:

1. כלל T-TRUE: ראינו בכיתה, אין צורך לכתוב.
2. כלל T-FALSE: ראינו בכיתה, אין צורך לכתוב.
3. כלל T-VAR: ראינו בכיתה, אין צורך לכתוב.
4. כלל T-ABS: ראינו בכיתה, אין צורך לכתוב.
5. כלל T-APP: ראינו בכיתה, אין צורך לכתוב.
6. כלל T-IF: השלימו.

תזכורת לכלל T-IF:

$$\frac{\Gamma\vdash t_1:Bool \quad \Gamma\vdash t_2:T \quad \Gamma\vdash t_3:T}{\Gamma\vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}\ (\text{T-IF})$$

- <u>חלק ב: מפרש לתחשיב למדא</u>

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

בתרגיל נעשה שימוש ב Ocaml. 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 ממומשות 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 2let x = t1 in t 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.

1. 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)

2. 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>

בס"ד

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 $y \in FV(s)$ when $z \notin FV(s) \cup FV(t) \cup FV(x)$
$(t_1\ t_2)[x \mapsto s]$	$= t_1[x \mapsto s]\ t_2[x \mapsto s]$	

3. 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.

4. 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.

5. הקובץ tests.ml מכיל את הפונקציה הבאה:

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

פונקציה זו מקבלת את אחת הפונקציות סעיפים 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.