

### שפות תכנות – תרגיל 3

**תאריך הגשה:** 08.01.2023

**הוראות הגשה:** ההגשה בזוגות. כל זוג נדרש לחשוב, לפתור ולכתוב את התרגיל בעצמו. מותר להתייעץ עם סטודנטים אחרים אך חל איסור מוחלט להחזיק ולהיעזר בתרגיל כתוב של סטודנט 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 מספר 6-7. קישור לתרגולים (המצגות נמצאות בתיאור הסרטון):

#### 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):



lexer.ml



parser.ml



readme.txt



reducer.ml



tests.ml



utils.ml



id.txt

בקובץ id.txt יש לכתוב את התז והשמות של המגשים.

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

**- חלק א: חישובים בתחשיב למדא****שאלה 1: חישוב ביטויי למדא**

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

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

ב.  $(\lambda x. \lambda y. x y y) (\lambda a. a) b$

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

הראו שיש שתי גזירות שונות (שמביאות גם לתוצאות שונות) לביטוי הבא:

ד.  $(\lambda x. y) ((\lambda y. y y y) (\lambda x. x x x))$

**שאלה 2: תחשיב למדא לביטויים בוליאניים**

נתונות ההגדרות הבאות (שחלקן ראינו בתרגול):

$$\text{tru} = \lambda t. \lambda f. t$$

$$\text{fls} = \lambda t. \lambda f. f$$

$$\text{test} = \lambda l. \lambda m. \lambda n. l m n$$

$$\text{or} = \lambda b. \lambda c. b \text{ True } c$$

א. כתבו חישוב (reduction) call-by-value semantics תחת לביטוי:

$$\text{test (or tru fls) a b}$$

כאשר a, b הם ערכים כלשהם.

ב. כתבו ביטוי בתחשיב למדא nand.

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

$$\text{nand tru fls}$$

$$\text{nand tru tru}$$

**שאלה 3:** תחשיב למדא לביטויים אריתמטיים

בתרגול ראינו את ההגדרות הבאות למספרים טבעיים ופעולות אריתמטיות:

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

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

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

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

...

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

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

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

$$iszero = \lambda m. m (\lambda x. False) True$$

- א. חשבו את  $succ c_0$  בעזרת Call-By-Name, האם התוצאה היא  $c_1$ ?
- ב. חשבו את  $succ c_0$  בעזרת Call-By-Value, האם התוצאה היא  $c_1$ ?
- ג. הגדירות פונקציית  $isodd$ , שתקבל מספר טבעי כפי שקודדנו בהגדרת השאלה ותחזיר  $tru$  אם המספר הוא אי-זוגי ו- $fls$  אם המספר זוגי.
- ד. חשבו את:

$isodd 3$

$isodd 4$

**שאלה 4:** Simply Typed Lambda Calculus

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

- א.  $f: Bool \rightarrow Bool \vdash (f (if true then false else true)): T$
- ב.  $f: Bool \rightarrow Bool \vdash (\lambda x: Bool. f (if x then false else true)): T$
- ג.  $\vdash (\lambda x: Bool. \lambda y: T. y x): Bool \rightarrow T \rightarrow Bool \rightarrow Bool$

## - חלק ב: מפרש לתחשיב למדא

בתרגיל זה נבנה Parser ו-Interpreter 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 ( $\lambda$ ):

$$t ::= id \mid (\backslash id. t) \mid (t_1 t_2) \mid (t) \mid \text{let } id = t_1 \text{ in } t_2$$

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. (\c. ((b c) fls))) in
((and tru) fls)
```

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

$$\text{Term} ::= id \mid \backslash id. \text{term} \mid \text{term1 term2}$$

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  $\text{let } x = t_1 \text{ in } t_2$  ע"י ייצוגם בתחביר האבסטרקטי כך:

$$(x. t_2) t_1$$

(השתכנעו שזהו אכן ייצוג שמשמר את המשמעות של  $\text{let expressions}$  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](http://OCaml%20library%3ASet.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:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x. t) &= FV(t) - \{x\} \\ FV(t_1 t_2) &= FV(t_1) \cup FV(t_2) \end{aligned}$$

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`.

הוסיפו לקובץ `reducer.ml` את הפונקציה:

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

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

$t_2 [x \rightarrow t_1]$

כלומר  $t_2$  כאשר כל המופעים של המשתנה  $x$  הוחלפו ב- $t_1$  (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:

$$\begin{aligned} x[x \mapsto s] &= s \\ y[x \mapsto s] &= y && \text{if } y \neq x \\ (\lambda x. t_1)[x \mapsto s] &= \lambda x. t_1 \\ (\lambda y. t_1)[x \mapsto s] &= \lambda y. t_1[x \mapsto s] && \text{if } y \neq x \text{ and } y \notin FV(s) \\ (\lambda y. t_1)[x \mapsto s] &= \lambda z. (t_1[y \mapsto z])[x \mapsto s] && \text{if } y \neq x \text{ and } y \in FV(s) \\ (t_1 t_2)[x \mapsto s] &= t_1[x \mapsto s] t_2[x \mapsto s] \end{aligned}$$

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:

$(\text{reduce\_cbv } t) = \text{Some } t'$  if  $t \rightarrow t'$  in call-by-value

$(\text{reduce\_cbv } t) = \text{None}$  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.