מושגים בשפות תכנות, שנה"ל תשע"ז, סמסטר א', מועד ב'

פרופ' מולי שגיב, עודד פדון

הנחיות כלליות:

- משך הבחינה 3 שעות.
- מותר להשתמש בכל חומר כתוב (אין להשתמש במחשבון, מחשב, או פלאפון).
- בכל השאלות עליכם לתת הסבר משכנע מדוע התשובה שכתבתם נכונה, אלא אם מצויין אחרת.
- במבחן 7 שאלות. שאלה 1 היא חובה, ועליכם לבחור לענות על 5 שאלות מבין שאלות 2-7.
- את כל התשובות יש לכתוב במחברת הבחינה בלבד, תשובות על טופס הבחינה לא ייבדקו.
- חלוקת הניקוד: שאלה 1, 10 נקודות, שאלות 2-7, 18 נקודות כל אחת. סה"כ 100 נקודות.

נא להקיף בעיגול את השאלות שבחרת לענות עליהן:

1 2 3 4 5 6 7

חובה לענות על השאלה הבאה:

שאלה 1 - שאלת חובה (10 נקודות - 2 נקודות לכל סעיף)

בשאלה זו עלייך לסמן נכון / לא נכון בכל אחד מהסעיפים (בשאלה זו אין צורך לנמק).

1. בשפת OCaml אפשר לשחרר את רשומת ההפעלה מהמחסנית בסוף פרוצדורה.
2. בשפת Go המתכנת משחרר זיכרון בצורה מפורשת.
3. אלגוריתם Hindley Milner להסקת טיפוסים תמיד מסתיים.
4. ב JavaScript אין High Order Functions.
5. התוכניות הבאות שקולות תחת Natural Operational Semantics:

- `while true do skip`
- `while true do skip ; x := 2`

בחרי 5 שאלות מהשאלות הבאות. משקל כל שאלה 18 נקודות.

שאלה 2 - Parsing

נתבונן בדקדוק הבא עבור שפה דמויית OCaml:

```
E ::= id | fun id "->" E | let id "=" E in E | E E | "(" E ")"
```

כאשר ה terminals הם:

```
id, fun, "->", let, "=", in, "(", ")"
```

וה non-terminal היחיד (וסמל ההתחלה) הוא E.

א. הראי שהדקדוק אינו חד משמעי ע"י מתן שני עצי גזירה שונים לאותה מילה.

ב. כתבי דקדוק חד משמעי שקול (עבור אותה שפה), כך שלהפעלה תהיה אסוציאטיביות שמאלית, וכן ביטויי fun וביטויי let ממשיכים ימינה ככל האפשר (להפעלה קדימות גבוהה משניהם), כמו בשפת OCaml. ציירי את עץ הגזירה בדקדוק שכתבת עבור המילה שבה השתמשת בסעיף א'. להלן דוגמאות לקדימות והאסוציאטיביות הרצויה:

הביטוי:

```
x y z
```

יתפרש בדומה לביטוי:

```
(x y) z
```

הביטוי:

```
fun x -> x y
```

יתפרש בדומה לביטוי:

```
fun x -> (x y)
```

והביטוי:

```
let x = z in x y
```

יתפרש בדומה לביטוי:

```
let x = z in (x y)
```

ג. האם הדקדוק שכתבת בסעיף ב' הוא LL(1)? אם לא, כתבי דקדוק LL(1) עבור אותה שפה. הדקדוק בסעיף זה לא חייב לשמור על האסוציאטיביות והקדימות של סעיף ב', אלא רק לגזור אותה שפה כמו הדקדוק מהסעיפים הקודמים.

ד. ציירי עץ גזירה לפי הדקדוק שכתבת בסעיף ג', עבור המילה:

```
let g = fun x -> x in g g
```

שאלה 3 - סמנטיקה

נתונות התוכניות הבאות בשפת While, כאשר b הוא ביטוי בוליאני כלשהו ו S,Q פקודות כלשהן:

P_1 = **while** b **do** S

P_2 = **while** b **do** (**if** b **then** S **else** Q)

א. כיצד מוגדרת שקילות סמנטית ב Natural Operational Semantics?

ב. האם התוכניות P_1 ו P_2 שקולות ב Natural Operational Semantics ? הוכיחי או תני דוגמה נגדית.

ג. כיצד מוגדרת שקילות סמנטית ב Structural Operational Semantics?

ד. האם התוכניות P_1 ו P_2 שקולות ב Structural Operational Semantics ? הוכיחי או תני דוגמה נגדית.

לנוחיותך, להלן שני משפטים שהוכחנו בקורס:

- אם $\langle S_1, s\rangle \Rightarrow^k s$ אז $\langle S_1;S_2, s\rangle \Rightarrow^k \langle S_2, s\rangle$
- אם $\langle S_1;S_2, s\rangle \Rightarrow^k \langle S_2, s''\rangle$ אז קיימים s', k_1, k_2 כך ש $k=k_1+k_2$ וכן:
  $\langle S_1, s\rangle \Rightarrow^{k1} s'$ וגם $\langle S_2, s'\rangle \Rightarrow^{k2} s''$

שאלה 4 - λ -calculus

נתון הביטוי הבא ב λ -calculus:

```
(λx. λy. y x) ((λx. x) (λx. x)) (λx. z)
```

א. ציירי את ה AST המתאים לביטוי.

ב. כתבי סדרת חישוב (reduction) לביטוי תחת call by value semantics.

ג. כתבי סדרת חישוב (reduction) לביטוי תחת lazy evaluation semantics.

ד. מצאי טיפוס T כך שקביעת הטיפוס הבא תתקיים, וכתבי עץ גזירה שמוכיח אותה לפי כללי הטיפוס (typing rules):

```
f:T ⊢ (λx:bool. f (f x)) : T
```

שאלה 5 - OCaml

א. נרצה לייצג ב OCaml ביטויים אריתמטיים מעל משתנים, כמו x+2*y או (x-y)*(x+y). לצורך כך, נגדיר טיפוס exp כדי לייצג את ה AST של הביטויים. השלימי את החלקים החסרים בהגדרת הטיפוס exp:

type exp = Id of char | Lit of float | Sum of _____ | Prod of _____

לאחר ההשלמה, ניתן לייצג את הביטוי x+2*y ע"י:

Sum (Id 'x', Prod (Lit 2.0, Id 'y'))

ב. כתבי פונקציה בשם eval מטיפוס exp -> (char -> float) -> float שתקבל ביטוי וסביבה, ותחשב את הערך של הביטוי בסביבה. הסביבה מיוצגת ע"י פונקציה char -> float שמחזירה את הערך של משתנה לפי שמו. לדוגמה, הקריאה הבאה:

```
let e = Sum (Id 'x', Prod (Lit 2.0, Id 'y')) in
let env = fun name -> if name='x' then 3.0 else 5.0 in
eval e env
```

תחזיר את הערך 13.0.

ג. כתבי פונקציה בשם derive מטיפוס exp -> char -> exp שתקבל ביטוי ושם של משתנה, ותחשב את הנגזרת של הביטוי לפי המשתנה.
לנוחיותך, כלל הנגזרת של מכפלה:

(f * g)' = f' * g + f * g'

שאלה 6 - JavaScript

ברצוננו לבנות אתר אינטרנט שיציג ספירה לאחור מ 10 עד 0 (שתימשך 10 שניות). בכדי להציג את הספירה, כתבנו את הפונקציה הבאה ב JavaScript:

```
function countdown() {
    var i;
    for (i = 10 ; i >= 0 ; i--) {
        alert(i); // show current value
        var t = this.now(); // returns current time in seconds
        while (this.now() < t + 1) {} // wait one second
    }
}
```

א. הסבירי בקצרה מה הבעיה במימוש הנ"ל, בהנחה שנטמיע אותו באתר האינטרנט שלנו.

ב. תקני את הקוד ע"י שימוש ב closure ובפונקציית ספריה מתאימה (אם אינך זוכרת את פונקציית הספריה המתאימה, המציאי עבורה שם ותארי במדויק מה היא עושה).

ג. הסבירי איך המימוש המתוקן שלך מסעיף ב' פותר את הבעיה מסעיף א'.

שאלה 7 - Types

נתון המימוש הבא של פונקציית fold_left ב OCaml:

```
let rec fold_left f acc lst = match lst with
    | [] -> acc
    | h :: t -> fold_left f (f acc acc) t
```

א. נתחי את הטיפוס של הפונקציה `fold_left` ע"י אלגוריתם Hindley-Milner. עלייך לפרט את אופן פעולת האלגוריתם עד להגעה לטיפוס הכללי ביותר של הפונקציה.

ב. כיצד ניתן להבין מהטיפוס שיש באג במימוש?

ג. תקני את הבאג.

ד. נתחי את הטיפוס של הפונקציה המתוקנת. עלייך להסביר בפירוט מה ישתנה בפעולת אלגוריתם Hindley-Milner על הקוד המתוקן לעומת סעיף א'.

להזכירכם: הפונקציה fold_left ב OCaml מקבלת שלושה פרמטרים: פונקציית עדכון, ערך התחלתי ל accumulator, ורשימה. תוצאת fold_left מחושבת ע"י הפעלת הפונקציה לעדכון ה accumulator עפ"י כל האיברים ברשימה, משמאל לימין. פורמלית:

fold_left f a [b1; ...; bn] = f (... (f (f a b1) b2) ...) bn

לדוגמה:

fold_left (-) 0 [2; 3; 4] = -9

בהצלחה,

מולי ועודד