מושגים בשפות תכנות, שנה"ל תשע"ה, סמסטר ב', 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)

בשאלה זו עלייך לסמן נכון / לא נכון בכל אחד מהסעיפים (בשאלה זו אין צורך לנמק). תשובה נכונה מזכה ב 2 נקודות לכל סעיף, ותשובה לא נכונה לא משפיעה על הציון.

1. בשפה שבה פונקציה יכולה להחזיר פונקציה כערך החזרה שלה, ויש וקינון סטטי של פונקציות, אין אפשרות לשחרר תמיד את רשומת ההפעלה של פונקציה כאשר הביצוע שלה מסתיים.
2. שפת Scala תומכת בחישובים עצלניים (lazy).
3. אם אין שימוש במשתנים לא לוקליים אז אין צורך ב access link.
4. שפת JavaScript היא Statically Typed.
5. התוכניות הבאות שקולות תחת Natural Operational Semantics:

- `x:=1 or (while true do skip)`
- `x:=1`

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

שאלה 2 - Parsing

נתבונן בדקדוק הבא עבור ביטויים בוליאניים שמכיל את האופרטור and והאופרטור not:

```
B → B and B | not B | ( B ) | id
```

כאשר B הוא non-terminal, וה terminals הם:

```
and, not, "(", ")", id
```

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 שלאופרטור not תהיה עדיפות על אופטור and, ולאופטור and תהיה אסוציאטיביות שמאלית. לדוגמה, המילה הבאה:

```
not x and not y and z
```

מתפרשת כ:

```
((not x) and (not y)) and z
```

ג. האם הדקדוק שרשמת בסעיף ב' הוא LL(1)? אם לא, כתבי דקדוק LL(1) שקול.

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

not not (x and y and z)

שאלה 3 - סמנטיקה

נרצה להוסיף לשפת While את הפקודה הבאה:

repeat odd S_1 **even** S_2 **until** b

זוהי לולאה שרצה לפחות פעם אחת ועד שהתנאי b מתקיים, באיטרציות אי-זוגיות מבצעת את S_1, ובאיטרציות זוגיות מבצעת את S_2 (האיטרציה הראשונה היא אי-זוגית). לדוגמה, התוכנית הבאה:

repeat odd n := n*2 **even** n := n*3 **until** n ≥ 8

תסתיים במצב בו n=12 בהנחה שהיא מתחילה ממצב בו n=1, ותסתיים במצב בו n=20 בהנחה שהיא מתחילה ממצב בו n=10.

א. הרחיבי את ה Natural Operational Semantics כדי לטפל בפקודה החדשה. הכללים אינם יכולים להסתמך על מבנה לולאת while הרגילה בשפה (כלומר לא ניתן להתייחס בכללים למבנה לולאת while הרגילה, אלא רק לפקודת הלולאה החדשה).

ב. הדגימי את הסמנטיקה המורחבת שהגדרת בסעיף א' ע"י בניית עץ גזירה לתוכנית מפסקת הפתיחה ממצב התחלתי בו n=1.

ג. הרחיבי את ה Structural Operational Semantics כדי לטפל בפקודה החדשה. הכללים אינם יכולים להסתמך על מבנה לולאת while הרגילה בשפה (כלומר לא ניתן להתייחס בכללים למבנה לולאת while הרגילה, אלא רק לפקודת הלולאה החדשה).

ד. הדגימי את הסמנטיקה המורחבת של סעיף ג' ע"י בניית סדרת גזירה לתוכנית מפסקת הפתיחה ממצב התחלתי בו n=1.

ה. האם ניתן לכתוב ביטוי בעזרת לולאת while רגילה ששקול סמנטית ללולאה החדשה? אם כן, כיצד? ואם לא, מדוע? (אין צורך להוכיח פורמלית, מספיק להסביר אינטואיטיבית).

שאלה 4 - λ-calculus

נתון הביטוי הבא ב λ-calculus:

$(\lambda t.\ \lambda f.\ t)\ (\lambda x.x)\ ((\lambda x.x)\ (\lambda s.\ \lambda z.\ s\ z))$

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. הסבירי בקצרה את ההבדל בין call by value ו lazy evaluation.

שאלה 5 - OCaml

הבהרה: בשאלה זו אסור להשתמש בפונקציות ספרייה מהמודול List

א. כתבי פונקציית OCaml בשם count שסופרת כמה פעמים איבר מופיע ברשימה. הנה מספר דוגמאות לשימוש בפונקציה:

```
# count 1 [0; 1; 3; 1; 2];;
- : int = 2
# count "ab" ["ab"; "ra"; "cad"; "ab"; "ra"];;
- : int = 2
# count None [Some 1; None; Some 2; None; None];;
- : int = 3
```

ב. הסיקי את הטיפוס הכללי ביותר של הפונקציה שכתבת בסעיף א'.

ג. מהי צריכת הזיכרון של הפונקציה שכתבת בסעיף א' כתלות באורך הרשימה שהיא מקבלת? הסבירי היכן זיכרון זה מוקצה, והאם ניתן להפעיל את הפונקציה על רשימה של מאה מיליון איברים?

ד. אם צריכת הזיכרון של הפונקציה שכתבת בסעיף א' אינה קבועה, כתבי גרסה שלה שיכולה לפעול בזיכרון קבוע ללא תלות באורך הרשימה (רמז: רקורסיית זנב). מותר להגדיר פונקציית עזר.

שאלה 6 - JavaScript

הפונקציה הבאה כתובה בצורה tail-recursive, היא מקבלת פונקציה f ומערך a, מפעילה את f על כל אחד מאיברי a ומחזירה את סכום התוצאות:

```
function mapreduce1(f, a) {
   function mr1(f, a, i, total) {
        if (i == a.length) return total;
        else return mr1(f, a, i+1, total + f(a[i]));
   }
   return mr1(f, a, 0, 0);
}
```

א. הסבירי כיצד ניתן להדר את הפונקציה mapreduce1 כך שבזמן הריצה, כמות הזיכרון שתצרוך הקריאה mapreduce1(f,a) תהיה בלתי תלויה בגודל המערך a.

ב. גרסה שונה, עדיין tail-recursive היא:

```
function mapreduce2(f, a) {
   function mr2(f, a, i, g) {
       if (i == a.length) return g(0)
       else return mr2(f, a, i+1, function(x) {return f(a[i]) + g(x);});
   }
   return mr2(f, a, 0, function(x) { return x; });
}
```

גם במקרה זה אין צורך לחזור מהקריאה הרקורסיבית ל mr2. אם כן, מדוע ההסבר שנתת בסעיף א' אינו תקף עבור הגרסה הזו, ומה תהיה צריכת הזיכרון של הקריאה mapreduce2(f,a) ?

ג. הקוד הבא משתמש בשתי הפונקציות שהוגדרו בסעיפים הקודימים:

```
function get_f() {
   var counter = 0;
   return function(x) {
       counter = counter + 1;
       return x * counter;
   };
}
var result1 = mapreduce1(get_f(), [1,2,3]);
var result2 = mapreduce2(get_f(), [1,2,3]);
```

לאחר הרצת הקוד הנ"ל, מה יהיו הערכים במשתנים result1 ו result2? הסבירי את תשובתך בעזרת טבלה שמתארת את שרשרת הקריאות הרקורסיביות בכל אחד מהמקרים.

שאלה 7 - Types

נתון המימוש הבא של פונקציית OCaml שאמורה לשרשר שתי רשימות:

```
let rec concat xs ys = match xs with
 | [] -> ys
 | x::xs' -> concat xs' ys
```

א. נתחי את הטיפוס של הפונקציה concat.

ב. כיצד ניתן להבין מהטיפוס שיש באג במימוש?

ג. תקני את הבאג.

ד. נתחי את הטיפוס של הפונקציה המתוקנת.

בהצלחה,

מולי ועודד