נתוח מתקדם של שפות תכנות
תרגיל 2
להגשה עד 12/12/2017

הנחיות כלליות:

- ניתן להגיש את התרגיל בזוגות.
- עבור Lambda Calculus, החומר הרלוונטי מופיע ב:
  Benjamin C. Pierce, Types and Programming Languages פרק 5.
- עבור סמנטיקה, החומר הרלוונטי מופיע ב:
  Glynn Winskel, Formal Semantics of Programming Languages פרק 2.

Lambda Calculus

1. נתונות ההגדרות הבאות (Church Booleans):

tru = λt. λf. t
fls = λt. λf. f
test = λl. λm. λn. l m n
and = λb. λc. b c fls

הגדירו ביטויים עבור הפונקציות הלוגיות xor, not ,or. הסבירו מדוע הביטויים שכתבת אכן ממשים את הפונקציות הלוגיות המתאימות.

2. נתונות ההגדרות הבאות (Church Numerals):

c_0 = λs. λz. z
c_1 = λs. λz. s z
c_2 = λs. λz. s (s z)
c_3 = λs. λz. s (s (s z))
… (c_k for any natural number k)
scc = λn. λs. λz. s (n s z)
plus = λm. λn. λs. λz. m s (n s z)
times = λm. λn. m (plus n) c_0

a. מצאו חישוב (reduction) תחת full-beta-reduction לביטוי scc c_0 . האם התוצאה שווה ל c_1?
b. מצאו חישוב (reduction) תחת call-by-value semantics לביטוי scc c_0 . האם התוצאה שווה ל c_1 ? באיזה מובן התוצאה שקולה ל c_1?
c. מצאו דרך אחרת להגדיר את scc, שתהיה עדיין פונקציית העוקב עבור Church Numerals הסבירו מדוע הביטוי שכתבתם עונה על הדרישה.
d. הגדירו פונקציה power להעלאת מספר בחזקה. הסבירו מדוע הביטוי שכתבתם עונה על הדרישה.
e. הגדירו פונקציה iszero, שתקבל Church Numeral ותחזיר Church Boolean, ותאפשר לבדוק האם מספר הוא אפס או לא. הסבירו מדוע הביטוי שכתבתם עונה על הדרישה.

3.

a. השתמשו ב Y-combinator כדי להגדיר פונקציה sum-l שתעבוד ב lazy evaluation semantics, ובהינתן Church Numeral עבור k, תחשב Church Numeral שמתאים לסכום כל המספרים הקטנים או שווים ל k. לדוגמה:

$sum\text{-}l\ c_0 \Rightarrow^* c_0$

$sum\text{-}l\ c_3 \Rightarrow^* c_6$ // 6 = 1 + 2 +3

$sum\text{-}l\ c_{10} \Rightarrow^* c_{55}$ // 55 = 1 + … + 9 + 10

$sum\text{-}l\ c_k \Rightarrow^* c_{1+...+k}$

לצורך הגדרת הפונקציה sum-l, ניתן להשתמש בכל הפונקציות שמופיעות בשאלות 3 ו- 4, וכן בפונקציה prd שמקיימת:

$prd\ c_0 \Rightarrow^* c_0$

$prd\ c_{k+1} \Rightarrow^* c_k$

הערה: אין צורך להגדיר את הפונקציה prd. ההגדרה המלאה שלה מופיעה בעמוד 62 בספר.

b. השתמשי ב Z-combinator ובפונקציה prd כדי להגדיר פונקציה sum-s שתעבוד ב ב call by value semantics ותקיים: $sum\text{-}s\ c_k \Rightarrow^* c_{1+...+k}$.

רמז: ב call by value semantics בעת הפעלת test על Church Boolean, גם ביטוי ה then וגם ביטוי ה else מחושבים - לכן, וודאי היטב שהפונקציה sum-s אכן מסתיימת!

Semantics

4. הוכיחו את השקילות הסמנטית הבאה:

(S1;S2);S3 ~ S1;(S2;S3)

כלומר, הוכיחו שלכל σ,σ' מתקיים:

$\langle (S1;S2);S3, \sigma\rangle \rightarrow \sigma'$ iff $\langle S1;(S2;S3), \sigma\rangle \rightarrow \sigma'$

5. נרצה להוסיף לשפת IMP את הפקודה הבאה:

repeat S **until** b

זוהי לולאה שתמיד מתבצעת פעם אחת לפחות, והביצוע שלה נפסק כאשר התנאי b מתקיים. לדוגמה, הקוד הבא:

repeat x := x-10 **until** x<10

יסתיים במצב בו x=5 אם יתחיל במצב בו x=55, ויסתיים במצב בו x=-3 אם יתחיל במצב בו x=7.

a. הרחיבו את התחבר של שפת IMP ואת הגדרת הסמנטיקה כדי שיגדירו את הסמנטיקה של קודת repeat. הכללים אינם יכולים להסתמך על מבנה לולאת while בשפה (כלומר לא ניתן להתייחס בכללים למבנה לולאת while, רק ללולאת repeat).

b. הוכיחו את השקילות הסמנטית הבאה ב IMP המורחבת שהגדרתם בסעיף a:

repeat S **until** b ~ S **; if** b **then** skip **else** (**repeat** S **until** b)

c. הוכיחו את השקילות הסמנטית הבאה ב IMP המורחבת שהגדרתם בסעיף a:

repeat S **until** b ~ S **; while** ¬b **do** S

6. בשפת IMP ביטויים הם side effect free, כלומר, לחישוב של ביטוי אין השפעה על המצב. נרצה להוסיף לשפת IMP ביטוי עם side effect, מהצורה:

c **resultis** a

כדי לשערך ביטוי זה, מבצעים את הפקודה c (מה שגורם ל side effects על המצב), ואז משערכים את a במצב החדש. לדוגמה, הביטוי:

x := x + 1 **resultis** x

מקביל לביטוי x++ בשפת C.

כדי לתמוך בשפה עם side effects, נגדיר מחדש את הסמנטיקה של ביטויים בדומה לסמנטיקה של פקודות, כך ש:

$$\langle a, \sigma \rangle \rightarrow \langle n, \sigma' \rangle$$

משמעו שחישוב הביטוי a במצב σ מסתיים עם ערך n, כאשר המצב השתנה (כתוצאה מ side effects) למצב 'σ.

הגדירו באופן פורמלי את התחביר והסמנטיקה של שפת IMP המורחבת באופן זה.

בהצלחה!