

שפות תכנות - 17/07/16, 17/07/16 - ק'

מחבר - ק'

אוניברסיטת אריאל בשומרון

שאלה 1 — BNF — (20 נקודות):

סעיף א' (6 נקודות):

כתבו דקדוק עבור שפת המחזורות הסופיות (באורך גדול מ-0) של אפסים ואחדות. להלן דוגמאות למילים חוקיות בשפה. הראו גזירה למילה באורך ארבעה תווים.

```
1 0 0 0
1 0 1 0 1 0 1 0
1 1
```

סעיף ב' (7 נקודות):

הסבירו את מושג ה-ambiguity ביחס לדקדוק שהגדרתם. בפרט הראו שני דקדוקים לאותה שפה (של סעיף א). אחד שמתקיימת בו התכונה ואחר שאינה מתקיימת בו. הסבירו את תשובתכם.

סעיף ג' (7 נקודות):

הסבירו את מושג ה-compositionality ביחס לדקדוק שהגדרתם. בפרט, הראו בחירה של סמונטיקה ושני דקדוקים לאותה שפה (של סעיף א). אחד שמתקיימת בו התכונה ואחר שאינה מתקיימת בו. הסבירו את תשובתכם.

שאלה 2 — With vs. Call — (20 נקודות):

סעיף א' (10 נקודות):

נתון הקוד הבא:

```
(run "{with { x { * 3 4 } }
      {with { foo { fun { y } { - x y } } }
        { call foo 2 } } }")
```

החליפו את הקוד הנ"ל בקוד שקול בו לא מופיעה המילה with - לצורך כך השתמשו במילה call.

סעיף ב' (10 נקודות):

סטודנט מחק את הוואריאנט (הבנאי) With של הטיפוס FLANG באינטרפרטר (במודל הסביבות) ואת כל שורות הקוד שהשתמשו בבנאי With. הכלילו את הרעיון מסעיף א', כדי לדאוג שהאינטרפרטר ימשיך לפעול כמצופה, גם ללא הבנאי With (אסור שהממשק מול מתכנת בשפה FLANG ישתנה).

הדרכה: החליפו את השורות המתאימות בפרוצדורה `parse-sexpr` - בשורות קוד שאינן מכילות הפעלה של `With` (אלא בנאים חלופיים קיימים). עליכם לכתוב במחברת רק את שורות הקוד המקוריות שבחרתם לשנות ומתחתיהן, את שורות הקוד החלופיות.

שאלה 3 — (37 נקודות): נחון הקוד הבא:

```
(run "{with {foo {fun {x} {* x x}}}  
      {call {with {foo {fun {y} {- y 1}}}  
              {fun {x} {call foo x}}}  
            4}}")
```

סעיף א' (15 נקודות):

תארו את הפעלות הפונקציה `eval` בתהליך ההערכה של הקוד מעלה במודל ה-`substitution-cache` (על-פי ה-`interpreter` התחתון מבין השלושה המצורפים מטה) - באופן הבא - לכל הפעלה מספר / תארו את AST_i - הפרמטר האקטואלי הראשון בהפעלה מספר i (עץ התחביר האבסטרקטי), את $Cache_i$ - הפרמטר האקטואלי השני בהפעלה מספר i (רשימת ההחלפות) ואת RES_i - הערך המוחזר מהפעלה מספר i . הסבירו בקצרה כל מעבר. ציינו מהי התוצאה הסופית.

דוגמת הרצה: עבור הקוד

```
(run "{with {x 1} {+ x 2}}")
```

היה עליכם לענות (בתוספת הסברים)

```
AST1 = (With x (Num 1) (Add (Id x) (Num 2)))  
Cache1 = '()  
RES1 = (Num 3)  
AST2 = (Num 1)  
Cache2 = '()  
RES2 = (Num 1)  
AST3 = (Add (Id x) (Num 2))  
Cache3 = '((x (Num 1))  
RES3 = (Num 3)  
AST4 = (Id x)  
Cache4 = '(x (Num 1))  
RES4 = (Num 1)  
AST5 = (Num 2)  
Cache5 = '((x (Num 1))  
RES5 = (Num 2)  
Final result: 3
```

✓ סעיף ב' (7 נקודות):

מה היה קורה לו היינו מבצעים את ההערכה במודל הסביבות? מהי התשובה הרצויה? מדוע? (אין צורך לבצע הערכה). תשובה מלאה לסעיף זה לא תהיה ארוכה משלוש שורות (תשובה ארוכה מדי תקרא חלקית).

✓ סעיף ג' (15 נקודות):

האינטרפרטר שכתבנו במודל ה-substitution-cache מממש dynamic-scoping ואילו האינטרפרטר שכתבנו במודל הסביבות מממש lexical-scoping. בשאלה זאת, עליכם לשנות את הקוד של האינטרפרטר שכתבנו במודל הסביבות (העליון מבין השניים המופיעים מטה) – על מנת להמירו למימוש של dynamic-scoping. עליכם לשנות מעט ככל שתוכלו מהקוד.

העתיקו למחברת רק את שורות הקוד שברצונכם לשנות וציינו בבירור מהו השינוי הנדרש. ככל שתשובתכם תכלול פחות שינויים לקוד המקורי, היא תזכה אתכם ביותר ניקוד. הסבירו בבירור מדוע השינויים שביצעתם נדרשים ומדוע הם מספיקים. תשובה ללא הסבר, לא תזכה אתכם בניקוד.

רמז: תשובה טובה תהיה קצרה מאד.

שאלה 4 – הרחבת השפה FLANG מודל הסביבות – (32 נקודות):

לצורך פתרון שאלה זו נעזר בקוד ה- interpreter של FLANG במודל הסביבות, המופיע בסוף טופס המבחן (העליון מבין השניים המופיעים שם).

נרצה להרחיב את השפה FLANG ולאפשר חישוב לולאות for עם פעולת חיבור או עם פעולת כפל. כלומר, נאפשר חזרה על קטע קוד, עם מונה (counter), כך שבחוק קטע הקוד מותר להשתמש בערך המונה (קטע הקוד עשוי להיות כל ביטוי על פי התחביר בשפה). הערך המוחזר מחישוב כל הלולאה יהיה סכום כל הערכים המוחזרים מכל החישובים בכל האיטרציות של הלולאה (אם הביטוי משתמש באופרטור +, ראו דוגמאות מטה) או מכפלת כל הערכים הללו (אם הביטוי משתמש באופרטור *).

להלן דוגמה לביטוי אפשרי:

```
"{ for + { i } = 1 to 12 do {+ 1 i} }"
```

כאן, i הוא שם המונה, 1 הוא ערך התחלתי למונה ו-12 הוא גבול עליון, {+ 1 i} הוא גוף הלולאה והפעולה היא פעולת החיבור (זאת על-פי השימוש באופרטור +). לשם פשטות, בהמשך ניתן תמיד להניח שערך הגבול העליון למונה גדול ממש מן הערך ההתחלתי עבור המונה וכי שניהם מספרים שלמים.

להלן דוגמאות למסמכים שאמורים לעבוד:

```
(test (run "{ for + { i } = 1 to 12 do 1 }")  
=> 12)
```

```
(test (run "{ for + { i } = {+ 1 0} to 5 do i }")
=> 15)
(test (run "{with {x { for + { i } = 1 to 5 do i }}
{- x 10}}")
=> 5)

(test (run "{ for * { i } = 1 to 4 do i }")
=> 24)

(test (run
"{with {factorial {fun {n} { for * { i } = 1 to n do i }}}
{call factorial 4}}")
=> 24)

(test (run "{ for + { i } = {with {x {- 5 3}} x} to {- 7 4} do
{with {sqr {fun {x} {* x x}}}
{call sqr i}}}")
=> 13)

(test (run
"{ + 5 { for + { i } = {with {x {- 5 3}} x} to {- 7 4} do
{with {sqr {fun {x} {* x x}}}
{call sqr i}}}}")
=> 18)

(test (run "{ for - { i } = 1 to 12 do 1 }")
=error> "parse-sexpr: bad `for' syntax in")
```

הערה: השתמשו בדוגמאות אלו בכדי להבין את דרישות התחביר, את אופן הערכת הקוד וכן את הודעות השגיאה שיש להדפיס במקרים המתאימים.

סעיף א' (הרחבת הדקדוק) (6 נקודות):

כתבו מהם שני כללי הדקדוק שיש להוסיף לדקדוק הקיים.

—«fill-in 1»—

—«fill-in 2»—

סעיף ב' (הרחבת הטיפוס FLANG) (5 נקודות):

הוסיפו את הקוד הנדרש (יש להוסיף בנאי יחיד) ל -

```
(define-type FLANG
...
[For —«fill-in 3»—]
...)
```

הדרכה: הארגומנט הראשון צריך להיות זה שיגדיר את הפעולה המתאימה (חיבור או כפל).

סעיף ג' (parsing) (10 נקודות):

השתמשו בדוגמאות המסמכים מעלה כדי להבין אילו הודעות שגיאה רצויות. הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים - 7 השלמות סה"כ לסעיף זה) ל -

```
(: parse-sexpr : Sexpr -> FLANG)
;; to convert s-expressions into FLANGs
(define (parse-sexpr sexpr)
  (match sexpr
    ...
    [(cons -«fill-in 4»-)
     (match -«fill-in 5»-
       [(list 'for -«fill-in 6»-)
        -«fill-in 7»-]
       [(list 'for -«fill-in 8»-)
        -«fill-in 9»-]
       [else -«fill-in 10»-]])]
    ...
    [else (error 'parse-sexpr "bad syntax in ~s"
                 sexpr)]))
```

סעיף ד' (evaluation) (11 נקודות):

בסעיף זה נשלים את הקוד שיאפשר הערכה של ביטויי לולאה כפי שהגדרנו בסעיפים הקודמים. ראשית, נהפוך את הפרוצדורה NumV->number זמינה ומוכרת (עד כה הייתה פנימית ל - arith-op).

```
(: NumV->number : VAL -> Number)
(define (NumV->number v)
  (cases v
    [(NumV n) n]
    [else (error ' NumV->number "expected a number, got: ~s" v)]))
(: arith-op : (Number Number -> Number) VAL VAL -> VAL)
;; gets a Racket numeric binary operator,
;; and uses it within a NumV wrapper
(define (arith-op op val1 val2)
  (NumV (op (NumV->number val1) (NumV->number val2))))
```

נתה נכתוב פונקציה שתקרא ע"י eval ותבצע את האיטרציות של הלולאה והחלת הפעולה האריתמטית על כל תוצאות החישובים האיטרטיביים. השלימו את הקוד החסר:

op - +, *

```
(: eval-for-loop : (Number Number -> Number) Symbol FLANG
FLANG FLANG ENV -> VAL)
(define (eval-for-loop op cnt-name from-exp to-exp body env)
  (: loop-eval : Number Number -> VAL)
  (define (loop-eval from to)
    (if -«fill-in 11»-
        (eval -«fill-in 12»-)
        (arith-op -«fill-in 13»-)
        (loop-eval -«fill-in 14»-)))
```

הדרכה: כדי להשלים את #11 - חישבו מהו תנאי העצירה המתאים במקרה שאמורה להיות איטרציה אחת. כדי להשלים את #12 - זכרו שבחוק גוף הלולאה ייחננו מופעים חופשיים של שם המונה.

הערה: לשם פשטות, ניתן להניח שערך הגבול העליון למונה גדול ממש מן הערך ההתחלתי עבור המונה וכי שניהם מספרים שלמים.

לבסוף, השלימו את שורת הקוד המחאימה בפונקציה eval:

```
(: eval : FLANG ENV -> VAL)
;; evaluates FLANG expressions by reducing them to values
(define (eval expr env)
  (cases expr
    ...
    [(For -«fill-in 15»-
      (-«fill-in 16»-))])))
```

-----<<<FLANG-ENV>>>-----

;; The Flang interpreter, using environments

#lang pl

#|

The grammar:

```
<FLANG> ::= <num>
          | { + <FLANG> <FLANG> }
          | { - <FLANG> <FLANG> }
          | { * <FLANG> <FLANG> }
          | { / <FLANG> <FLANG> }
          | { with { <id> <FLANG> } <FLANG> }
          | <id>
          | { fun { <id> } <FLANG> }
          | { call <FLANG> <FLANG> }
```

Evaluation rules:

eval(N,env)	= N
eval({+ E1 E2},env)	= eval(E1,env) + eval(E2,env)
eval({- E1 E2},env)	= eval(E1,env) - eval(E2,env)
eval({* E1 E2},env)	= eval(E1,env) * eval(E2,env)

• (1c)

1 (3) 1 <bits>

1 (4) 0 <bits>

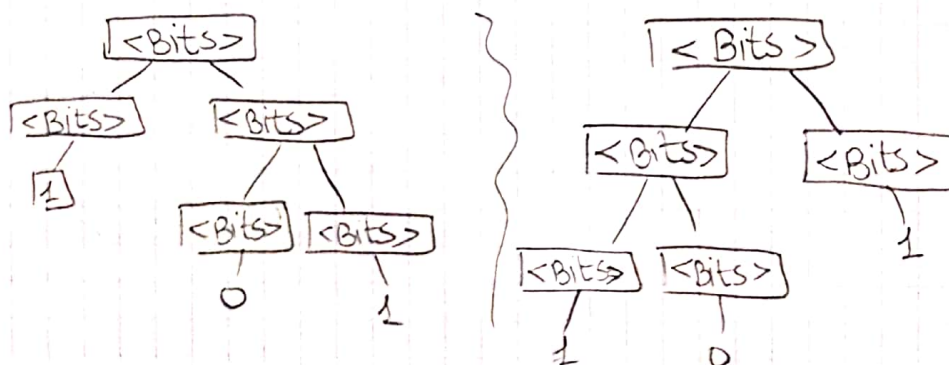
$$\begin{array}{ccccccc} & & & & 1010 & & - \text{off N} \\ & & & & & & \\ \langle \text{Bits} \rangle^{(3)} & \Rightarrow & 1 & \langle \text{Bits} \rangle^{(4)} & \Rightarrow & 10 & \langle \text{Bits} \rangle^{(3)} \Rightarrow 101 \\ \stackrel{(2)}{\Rightarrow} & 1010 & & & & & \end{array}$$

(ב). ambiguity - שתי מצבים של 13-מסמך, סומה (נ)
 חסר מילה במפה 2-38 ימים.
 הדקדוק שלשני הסוף תל - הי סוף 13-מסמך.
 (ואה) דקדוק שבו ה- ambiguity

1 (2) 1

1 (3) $\langle \text{Bits} \rangle \quad \langle \text{Bits} \rangle$

მის- 36 2 ასე- 101 - 7128 სენატის



Compositionality - אלוהי שפת תחביר (Syntax) (ד)

האם תמיד נכון להניח

האם תמיד נכון להניח

$$\langle \text{Bits} \rangle ::= 1 \mid 0$$

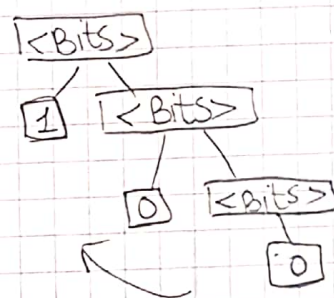
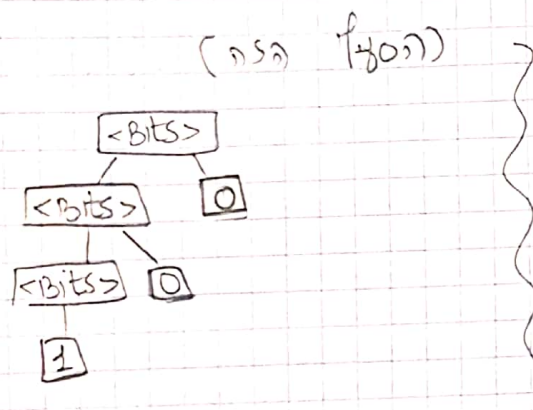
$$\mid \langle \text{Bits} \rangle 1$$

$$\mid \langle \text{Bits} \rangle 0$$

האם תמיד נכון להניח

האם תמיד נכון להניח

האם תמיד נכון להניח



האם תמיד נכון להניח

$$1 \cdot 2^2 + 0 + 0 = 4$$

האם תמיד נכון להניח

האם תמיד נכון להניח

$$\text{eval}(x) = x + 2 \cdot \text{eval}(\text{left})$$

$$0 + 2 \cdot \text{eval}(\text{left}) = 2 \cdot (0 + 2 \cdot \text{eval}(\text{left})) =$$

$$= 2 \cdot (2 \cdot 1) = 4$$

Compositionality

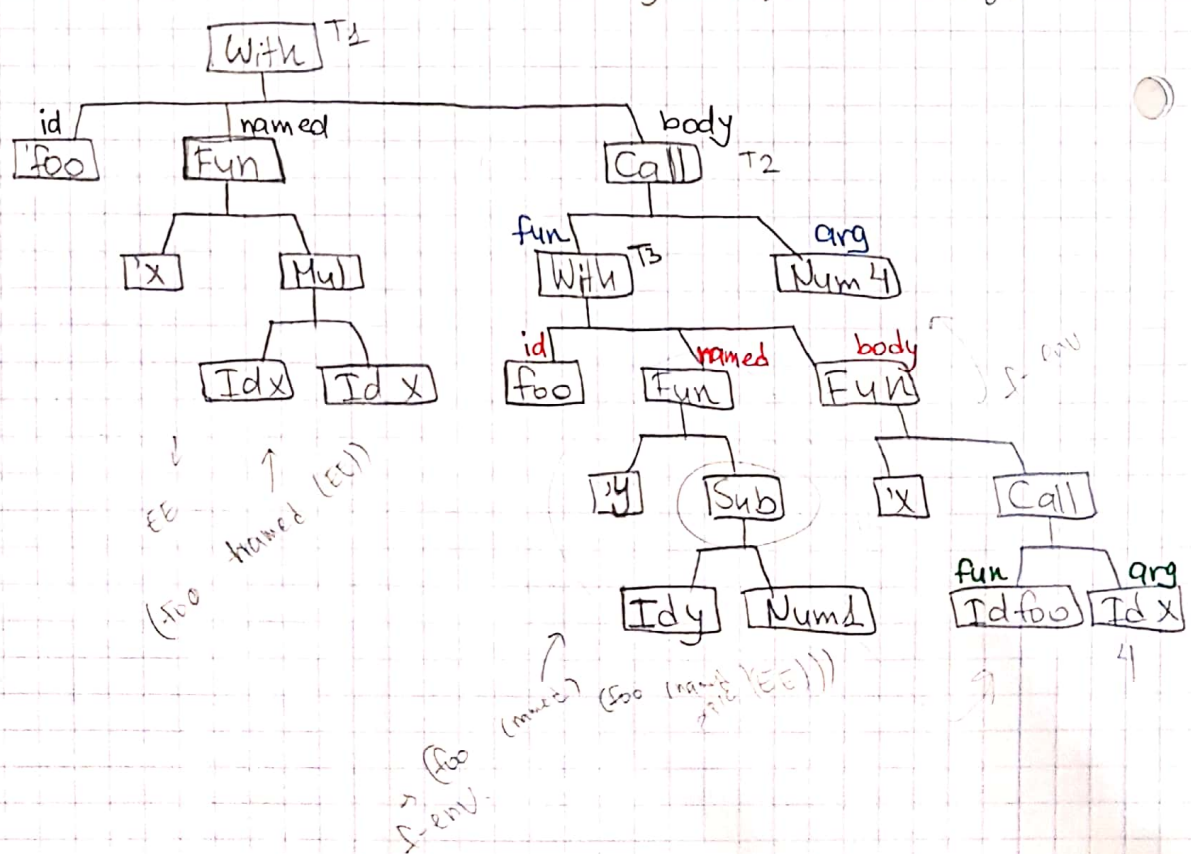
(With { id named } body) - 12/10/20 . (12)
 (Call (Fun id body) named)

{call {fun foo {call foo 2}}
 body {fun {y} {- x y}}}

(run "{call {fun {x}
 {call {fun {foo} {call foo 2}}
 {fun {y} {- x y}}}
 {x 3 4}}}")

- parse-sexpr - 12/10/20 . (2)
 (Call (Fun name (parse-sexpr body)) (parse-sexpr named))

- 12/10/20 12/10/20



AST1: T1

(Sc) . (lc)

Cache1: '()

Res1: (Num 16)

AST2: (Fun x (Mul (Idx) (Id x))) } eval(named)

Cache2: '()

Res2: (Fun x (Mul (Idx) (Id x)))

AST3: T2 eval(body)

Cache3: ((foo (Fun x (Mul (Idx) (Id x)))))

Res3: Res = (Num 16)

AST4: T3

eval(fun)

Cache4: Cache 3

Res4: Res 6.

AST5: (Fun y (Sub (Id y) (Num 1))) eval(named)

Caches: Cache 4

Res5: (Fun y (Sub (Id y) (Num 1)))

AST6: (Fun x (Call (Id foo) (Id x))) eval(body)

Cache6: ((foo (Fun y (Sub (Id y) (Num 1)))) Cache 4)

Res6: (Fun ^x_{id} (Call (Id foo) (Id x)))
_{body}

AST7: (Num 4)

eval(arg)

Cache7: Cache 3

Res7: (Num 4)

AST8: (Call ^{fun} (Id foo) ^{arg} (Id x)) eval(body)

Caches: ((x (Num 4)) Cache 3)

Res8: (Num 16) = Res 11

(3)

eval(foo)

AST9: (Id foo)

Cache9: Cache8

Res9: (Fun x (Mul (Id x) (Id x)))

eval(arg)

AST10: (Id x)

Cache10: Cache8

Res10: (Num 4)

AST11: (Mul (Id x) (Id x))

Cache11: ((x (Num4)) Cache8)

Res11: (Num 16)

AST12: (Id x)

Cache12: Cache11

Res12: (Num 4)

AST13: (Id x)

Cache13: Cache11

Res13: (Num 4)

(ב). במקרה הסדרה הינו מקדים - 3, משום שהפונקציה

"שומרת" על הסביבה בה הן הוגדרו והמשקה שאנו רוצים

הוא 3 - משום שאנו רוצים lexical scoping (לא dynamic scoping)

(ג). כדי לשנות dynamic לא נכנסה משנה אלא הסדרה

בה הפונקציה הוגדרה, אלא שההסדרה שאנו קוראים לה -

ואכן נשנה כן - (אופרה מה - Call) במקרה - let - ק - Cases

[(FunV bound-id bound-body env) ...]

זה הפונקציה, זהו! (U)

1. $1\{ \text{for } + \{ \langle \text{id} \rangle \} = \langle \text{FLANG} \rangle \text{ to } \langle \text{FLANG} \rangle \text{ do } \langle \text{FLANG} \rangle \}$. (1c)

2. $1\{ \text{for } * \{ \langle \text{id} \rangle \} = \langle \text{FLANG} \rangle \text{ to } \langle \text{FLANG} \rangle \text{ do } \langle \text{FLANG} \rangle \}$

3. [For (U '+'*) Symbol FLANG FLANG FLANG] . (2)

4. (cons 'for more)

5. (match sexpr

6. [list 'for '+' (list (symbol: counter))
'= num1 'to num2 'do calc)

7. (For '+' counter (parse-sexpr num1)
(parse-sexpr num2)
(parse-sexpr calc))]

8. [(list 'for '*' (list (symbol: counter))
'= num1 'to num2 'do calc)

9. (For '*' counter (parse-sexpr num1)
(parse-sexpr num2)
(parse-sexpr calc)))]

10. [else (error 'parse-sexpr "bad 'for' syntax in ~s" sexpr)]

11. (if (= from to) . (a))

12. (eval body

(Extend cnt-name (NumV from) env))

13. (arith-op op (eval body (Extend cnt-name (NumV from) env))

(loop-eval (add 1 from) to))

14. (loop-eval (NumV → number (eval from-exp env))
(NumV → number (eval to-exp env)))

15. [(For op counter from to body)

16. (eval-for-loop op counter from to body env)]