אוניברסיטת
אריאל
בשומרון

פקולטה: מדעי הטבע

מחלקה: מדעי המחשב

שם הקורס: שפות תכנות

מספר הקורס: 2-7036010

קבוצות: 1, 2, 4, 5

תאריך בחינה: 11/07/2022 סמ' ב' מועד א'

משך הבחינה: שעתיים וחצי

שם המרצה: מר' סעיד עסלי

מתרגלים: מר' עידו שפירא, מר' רועי פלג.

חומר עזר: אין.

שימוש במחשבון: לא.

הוראות כלליות:

- כתבו את תשובותיכם בכתב קריא ומרווח.
- בכל שאלה או סעיף, ניתן לכתוב – **לא יודע/ת** (מבלי להוסיף דבר מעבר לכך) – ולקבל 20% מהניקוד על השאלה או הסעיף.
- אפשר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם.
- יש לענות על כל השאלות.

אוניברסיטת
אריאל
בשומרון

שאלה 1 — BNF — (20 נקודות):

נתון הדקדוק (BNF) הבא:

```
<ME>  ::= <num>
        | <ME> * <ME>
        | <ME> / <ME>
```

כאשר `<num>` מתאר ערך מספרי כלשהו על-פי הגדרת RACKET.

סעיף א' (5 נקודות):

מהי השפה שמגדיר הדקדוק? ציירו עץ גזירה עבור מילה שבה מופיעים לפחות ארבעה מספרים (כל מספר כזה צריך להיות בן שתי ספרות – שהן צמד ספרות מתוך מספר ת.ז. של כותב המבחן).

סעיף ב' (5 נקודות):

הדקדוק הנתון אינו חד-משמעי (כלומר, הוא סובל מ-ambiguity). הראו זאת. הסבירו מדוע זו בעיה בהקשר של שפות תכנות (השתמשו בדקדוק הנתון, כדי להדגים את הבעיה והסבירו מתי היא תתעורר).

סעיף ג' (5 נקודות):

בסעיף זה תהפכו את הדקדוק לחד-משמעי מבלי לשנות את השפה. על הדקדוק החדש שתכתבו לקיים בפרט:

- **"4 * 3 / 12" – היא מילה קבילה בשפה, וגם**
- **אם נעריך את "4 * 3 / 12" כך שנפרש את סימן * כפעולת כפל ואת סימן / כפעולת חילוק, אזי הערך שיתקבל יהיה: 16.**

סעיף ד' (5 נקודות):

הסבירו כיצד נפתרה הבעיה שהדגמתם בסעיף ב (כתבו הסבר קצר במשפט או שניים – הסבר ארוך לא יתקבל).

אוניברסיטת
אריאל
בשומרון

שאלה 2 — Functions as first class — (30 נקודות):

בכיתה דיברנו על כך שבשפה racket כמו גם בשפה שלנו FLANG -- ההתייחסות לפונקציות היא כ-first class.

בשאלה זאת נדון במשחק שבו מטילים n מטבעות (ערך המטבע הינו או 0 או 1) ובודקים מה מספר האחדות שיצאו. למעשה, נכתוב פונקציה המחשבת את ההסתברות לקבל k אחדות מתוך n המטבעות (לחלופין, n-k אפסים).

תזכורת: מטבע נקרא p-biased אם כשמטילים אותו הוא מקבל ערך 1 בהסתברות p וערך 0 בהסתברות 1-p. ההסתברות לקבלת k אחדות מתוך n הטלות בלתי תלויות של מטבע שהוא p-biased נתונה על-ידי הנוסחה הבאה:

$$\frac{n!}{k! \cdot (n-k)!} \cdot p^k \cdot (1-p)^{n-k}$$

סעיף א' – פונקצית עזר לחישוב המקדם הבינומי – (10 נקודות):

כתבו פונקציה `choose` אשר בהנתן קלט של שני טבעיים n, k, מחשבת את המקדם הבינומי המוגדר על-ידי הביטוי $\frac{n!}{k! \cdot (n-k)!}$. במילים אחרות (`choose` $k\ n$) מחשבת את מספר האפשרויות לבחור קבוצה בגודל k מתוך קבוצה בגודל n. תוכלו להניח כי קיימת פונקציה בשם fact שמקבלת מספר טבעי ומחזירה את העצרת שלו.

סעיף ב' – יצירת פונקציה המממשת מטבע p-biased – (15 נקודות):

כתבו פונקציה `createBiasedCoin` אשר מקבלת כקלט מספר $0 \le p \le 1$ ומחזירה כפלט פונקציה משני מספר טבעיים n, k למספר ממשי – בהנתן קלט n, k הפונקציה (שהיא התוצר של `createBiasedCoin`) מחשבת את ההסתברות לקבלת k אחדות מתוך n הטלות בלתי תלויות של מטבע p-biased. לצורך כך, תוכלו להשתמש בפונקציה `expt` של `racket`, אשר לוקחת a ו-b ומחזירה a^b. עוד על פעולתה של `expt` בהמשך טופס הבחינה.

הערות טכניות:

- **יש לבדוק שמתקיים $0 \le p \le 1$.**
- **תוכלו להשתמש בפונקציה שכתבתם בסעיף הקודם.**

להלן דוגמאות הרצה (מול סביבת העבודה של DrRacket):

```
> (createBiasedCoin 1/2)
- : (Nonnegative-Integer Nonnegative-Integer -> Number)
#<procedure:coinP>
> (createBiasedCoin 2/2)
- : (Nonnegative-Integer Nonnegative-Integer -> Number)
#<procedure:coinP>
> (createBiasedCoin 3/2)

createBiasedCoin: expected a probability value, got 3/2
```

```
(define c-half (createBiasedCoin 1/2))
(define c-2thirds (createBiasedCoin 2/3))
(test (c-half 2 4) => 3/8)
(test (c-half 3 4) => 1/4)
(test (c-2thirds 2 4) => 8/27)
(test (c-2thirds 3 4) => 32/81)
(test (c-2thirds 1 3) => (* 3 2/3 (expt 1/3 2)))
```

<u>שימו לב</u>:

1. `createBiasedCoin` אמורה להחזיר פונקציה משני טבעיים למספר.
2. לצורך כתיבתה תוכלו להשתמש בקוד החלקי הבא (מותר לכם גם לכתוב קוד אחר עבור אותה מטרה).

```
(: createBiasedCoin : Number ->  <--fill in 1-->)
(define (createBiasedCoin p)
  (: k-ones-of-n : Natural Natural -> Number)
  ;; Computes the probability of exactly k ones
  (define (k-ones-of-n  k n)
    <--fill in 2-->)
  (: coinP : Natural Natural -> Number)
  (define (coinP K N)
    <--fill in 3-->)
  (if (or <--fill in 4-->)
      (error 'createBiasedCoin <--fill in 5-->)
     <--fill in 6-->))
```

<u>רמז</u>: `k-ones-of-n` היא פונקצית עזר המחשבת את ההסתברות הרצויה הלכה למעשה לאחר שניתנים לה p והפרמטרים k,n.
הפונקציה `coinP` מכירה את `k-ones-of-n`.

<u>סעיף ג' – (5 נקודות):</u> הסבירו בלכל היותר שלושה משפטים, מה הכוונה ששפה מתייחסת לפונקציות כ-first class וכיצד זה בא לידי ביטוי בקוד שכתבתם.

<u>שאלה 3 — (20 נקודות):</u>

נתון הקוד הבא:

```
(run "{with {x 3}
      {with {f {fun {y} {+ x y}}}
            {with {x 5}
                  {call f x}}}}")
```

הסבירו מה היה קורה אם היינו מבצעים הערכה של הקוד מעלה במודל ה- substitution cache – יש צורך בחישוב מלא (הסבירו את כל השלבים). מהי התשובה שעליה החלטנו בקורס כרצויה? מדוע?

<u>שאלה 4 — הרחבת השפה FLANG – טיפול בהטלות מטבע מוטה — (30 נקודות):</u>

לצורך פתרון שאלה זו נעזר בקוד ה- interpreter של FLANG במודל הסביבות, המופיע בסוף טופס המבחן. למעשה, נרחיב אותו על-מנת לאפשר שימוש בביטויים ופעולות (כגון חישוב הסתברויות) על מטבע עם הטיה p. להלן דוגמאות לטסטים שאמורים לעבוד:

```
;; tests
(test (parse "{coin 1/23}") => (Coin (Num 1/23)))
                                    שימו לב שמעלה הופעלה parse

(test (run "{with {c-half {coin 1/2}}
               {coinProbKN c-half 2 4}}") => 3/8)
(test (run "{with {c-half {coin {/ 1 {+ 1 1}}}}
               {coinProbKN c-half 3 4}}") => 1/4)
(test (run "{with {x 1/5}
          {with {c {coin x}}
             {with {x 1/2}
               {coinProbKN c 2 4}}}}")
      => 96/625)
(test (run "{coinProbKN {coin 1/2} 2 4}") => 3/8)
(test (run "{with {x 1/3}
              {with {c {coin x}}
                {+ {coinProbKN {coin 1/2} 2 4}
                   {coinProbKN c 2 4}}}}")
      => 145/216)
(test (run "{with {x 1/3}
            {with {c {coin x}}
```

```
                {coinProbKN c 2 4}}}")
      => 8/27)
(test (run "{with {x 1/3}
               {with {c {coin x}}
                 {coinProbKN c {with {c 5} {+ c 4}} 22}}}")
      => 4074864640/31381059609)
(test (run "{coin 1/2}") =error> "run: evaluation returned a
non-number: (CoinV #<procedure:coinP>)")
```

רוב הקוד הנדרש ניתן לכם מטה ואתם נדרשים להשלים רק את הדקדוק, הרחבת FLANG ואת הרחבת ה-eval. למען הקיצור, מובאות בפניכם רק התוספות לקוד הקיים. (שלוש נקודות "..." מופיעות במקום הקוד המושמט). הקוד לשימושכם, אך אין צורך להתעמק בכולו. השתמשו רק בחלקים שחשובים לכם לפתרון סעיפים א' ו-ב', הנתונים מטה.

```
  ;; The FLANG interpreter with biased coins, using
environments

  #lang pl

  #|
  The grammar:
```

<u>סעיף א' — הרחבת הדקדוק והטיפוס FLANG— (10 נקודות):</u>

לצורך הרחבת הדקדוק ישנם שני כללים חדשים. השלימו את החלקים החסרים בדקדוק הבא ובהגדרת FLANG מטה על פי דוגמאות ההרצה מעלה (תוכלו גם להעזר בשאר הקוד הנתון לכם).

```
    <FLANG> ::=
                 …
                | { coin <--fill in 1--> }
                | { coinProbKN <--fill in 2--> }

  (define-type FLANG
   …
    [Coin <--fill in 3-->]
    [CoinPrKN <--fill in 4-->])

  (: parse-sexpr : Sexpr -> FLANG)
  ;; to convert s-expressions into FLANGs
  (define (parse-sexpr sexpr)
    (match sexpr
  …
    [(list 'coin pr) (Coin (parse-sexpr pr))]
    [(list 'coinProbKN p k n) (CoinPrKN (parse-sexpr p)
                                        (parse-sexpr k)
                                        (parse-sexpr n))]
      [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))
```

אוניברסיטת
אריאל
בשומרון

סעיף ב' — eval — (20 נקודות):

בסעיף זה, עליכם לכתוב את הפונקציה eval עצמה. שימו לב להגדרות הבאות ולהרחבה לטיפוס VAL.

```
Evaluation rules:
…
 eval({coin E} ,env)       = (makeCoin (eval E env))
 eval({coinProbKN E1 Ek En},env)
      = probability of K ones in N sample of a p-biased coin
             by : (c K N)
               c = (eval E1 env) implements a p-biased coin,
               K = (eval Ek env),
               N = (eval En env).
|#
(define-type VAL
  [NumV Number]
  [CoinV (Natural Natural -> Number)]
  [FunV Symbol FLANG ENV])
```

כמו כן, הפונקציה הבאה תשמש אתכם במקומות אחרים ולכן, תהיה חיצונית (גלובאלית).

```
 (: NumV->number : VAL -> Number)
  (define (NumV->number v)
    (cases v
      [(NumV n) n]
      [else (error 'eval "expected a number, got: ~s" v)]))
```

כתבו את eval. הוסיפו את הקוד הנדרש (היכן שכתוב –«fill-in»–).

- **הניחו שישנה פונקצית המרת טיפוס `Number->Natural` הלוקחת מספר וממירה אותו לטבעי.**

```
(: eval : FLANG ENV -> VAL)
;; evaluates FLANG expressions by reducing them to values
(define (eval expr env)
  (cases expr
    …
    [<--fill in 5-->]
    [<--fill in 6-->
         [else (error 'eval "`coinProbKN' expects a coin,
            got: ~s" c-val)])]))
```

אוניברסיטת
אריאל
בשומרון

הדרכה: תוכלו להשתמש בפונקציות שכתבתם בשאלות קודמות (בפרט, בסעיף ב', שאלה 2).
בהשלימכם את הקוד מעלה, ודאו שאתם מקפידים על הטיפוס הנכון של אובייקט הנשלח כארגומנט לפרוצדורה אחרת או כערך מוחזר לחישוב הנוכחי.

תזכורות ואינטרפרטרים לשימושכם:

תזכורת: (expt)

```
Procedure
(expt z w) → number?

  z : number?

  w : number?
Returns z raised to the power of w.
Examples:
> (expt 2 3)
8
> (expt 4 0.5)
2.0
> (expt +inf.0 0)
1
```

```
--<<<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)
    eval({/ E1 E2},env)        = eval(E1,env) / eval(E2,env)
    eval(x,env)                = lookup(x,env)
    eval({with {x E1} E2},env) = eval(E2,extend(x,eval(E1,env),env))
    eval({fun {x} E},env)      = <{fun {x} E}, env>
    eval({call E1 E2},env1)
             = eval(Ef,extend(x,eval(E2,env1),env2))
                               if eval(E1,env1) = <{fun {x} Ef}, env2>
             = error!          otherwise
```

```
|#
(define-type FLANG
  [Num  Number]
  [Add  FLANG FLANG]
  [Sub  FLANG FLANG]
  [Mul  FLANG FLANG]
  [Div  FLANG FLANG]
  [Id   Symbol]
  [With Symbol FLANG FLANG]
  [Fun  Symbol FLANG]
  [Call FLANG FLANG])
(: parse-sexpr : Sexpr -> FLANG)
;; to convert s-expressions into FLANGs
(define (parse-sexpr sexpr)
  (match sexpr
    [(number: n)     (Num n)]
    [(symbol: name) (Id name)]
    [(cons 'with more)
     (match sexpr
       [(list 'with (list (symbol: name) named) body)
        (With name (parse-sexpr named) (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `with' syntax in ~s" sexpr)])]
    [(cons 'fun more)
     (match sexpr
       [(list 'fun (list (symbol: name)) body)
        (Fun name (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `fun' syntax in ~s" sexpr)])]
    [(list '+ lhs rhs) (Add (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '- lhs rhs) (Sub (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '* lhs rhs) (Mul (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '/ lhs rhs) (Div (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list 'call fun arg) (Call (parse-sexpr fun) (parse-sexpr arg))]
    [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))
(: parse : String -> FLANG)
;; parses a string containing a FLANG expression to a FLANG AST
(define (parse str)
  (parse-sexpr (string->sexpr str)))

;; Types for environments, values, and a lookup function
(define-type ENV
  [EmptyEnv]
  [Extend Symbol VAL ENV])
(define-type VAL
  [NumV Number]
  [FunV Symbol FLANG ENV])
(: lookup : Symbol ENV -> VAL)
(define (lookup name env)
  (cases env
    [(EmptyEnv) (error 'lookup "no binding for ~s" name)]
    [(Extend id val rest-env)
     (if (eq? id name) val (lookup name rest-env))]))

(: 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->number : VAL -> Number)
    (define (NumV->number v)
      (cases v
        [(NumV n) n]
        [else (error 'arith-op "expects a number, got: ~s" v)]))
    (NumV (op (NumV->number val1) (NumV->number val2))))

  (: eval : FLANG ENV -> VAL)
  ;; evaluates FLANG expressions by reducing them to values
  (define (eval expr env)
    (cases expr
      [(Num n) (NumV n)]
      [(Add l r) (arith-op + (eval l env) (eval r env))]
      [(Sub l r) (arith-op - (eval l env) (eval r env))]
      [(Mul l r) (arith-op * (eval l env) (eval r env))]
      [(Div l r) (arith-op / (eval l env) (eval r env))]
      [(With bound-id named-expr bound-body)
       (eval bound-body
             (Extend bound-id (eval named-expr env) env))]
      [(Id name) (lookup name env)]
      [(Fun bound-id bound-body)
       (FunV bound-id bound-body env)]
      [(Call fun-expr arg-expr)
       (let ([fval (eval fun-expr env)])
         (cases fval
           [(FunV bound-id bound-body f-env)
            (eval bound-body
                  (Extend bound-id (eval arg-expr env) f-env))]
           [else (error 'eval "`call' expects a function, got: ~s"
                        fval)]))]))
  (: run : String -> Number)
  ;; evaluate a FLANG program contained in a string
  (define (run str)
    (let ([result (eval (parse str) (EmptyEnv))])
      (cases result
        [(NumV n) n]
        [else (error 'run
                     "evaluation returned a non-number: ~s" result)])))
```

```
--<<<FLANG-Substitution-cache>>>-----------------------------------------
  ;; The Flang interpreter, using Substitution-cache
#lang pl
  (define-type FLANG
    [Num  Number]
    [Add  FLANG FLANG]
    [Sub  FLANG FLANG]
    [Mul  FLANG FLANG]
    [Div  FLANG FLANG]
    [Id   Symbol]
    [With Symbol FLANG FLANG]
    [Fun  Symbol FLANG]
    [Call FLANG FLANG])
```

```
(: parse-sexpr : Sexpr -> FLANG)
;; to convert s-expressions into FLANGs
(define (parse-sexpr sexpr)
  (match sexpr
    [(number: n)     (Num n)]
    [(symbol: name) (Id name)]
    [(cons 'with more)
     (match sexpr
       [(list 'with (list (symbol: name) named) body)
        (With name (parse-sexpr named) (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `with' syntax in ~s" sexpr)])]
    [(cons 'fun more)
     (match sexpr
       [(list 'fun (list (symbol: name)) body)
        (Fun name (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `fun' syntax in ~s" sexpr)])]
    [(list '+ lhs rhs) (Add (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '- lhs rhs) (Sub (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '* lhs rhs) (Mul (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '/ lhs rhs) (Div (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list 'call fun arg) (Call (parse-sexpr fun) (parse-sexpr arg))]
    [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))

(: parse : String -> FLANG)
;; parses a string containing a FLANG expression to a FLANG AST
(define (parse str)
  (parse-sexpr (string->sexpr str)))

;; a type for substitution caches:
(define-type SubstCache = (Listof (List Symbol FLANG)))
(: empty-subst : SubstCache)
(define empty-subst null)

(: extend : Symbol FLANG SubstCache -> SubstCache)
(define (extend name val sc)
  (cons (list name val) sc))

(: lookup : Symbol SubstCache -> FLANG)
(define (lookup name sc)
  (let ([cell (assq name sc)])
    (if cell
      (second cell)
      (error 'lookup "no binding for ~s" name))))

(: arith-op : (Number Number -> Number) FLANG FLANG -> FLANG)
;; gets a Racket numeric binary operator, and uses it within a FLANG
;; `Num' wrapper
(define (arith-op op expr1 expr2)
  (: Num->number : FLANG -> Number)
  (define (Num->number e)
    (cases e
      [(Num n) n]
      [else (error 'arith-op "expects a number, got: ~s" e)]))
  (Num (op (Num->number expr1) (Num->number expr2))))
```

```
(: eval : FLANG SubstCache -> FLANG)
;; evaluates FLANG expressions by reducing them to expressions
(define (eval expr sc)
  (cases expr
    [(Num n) expr]
    [(Add l r) (arith-op + (eval l sc) (eval r sc))]
    [(Sub l r) (arith-op - (eval l sc) (eval r sc))]
    [(Mul l r) (arith-op * (eval l sc) (eval r sc))]
    [(Div l r) (arith-op / (eval l sc) (eval r sc))]
    [(With bound-id named-expr bound-body)
     (eval bound-body
           (extend bound-id (eval named-expr sc) sc))]
    [(Id name) (lookup name sc)]
    [(Fun bound-id bound-body) expr]
    [(Call fun-expr arg-expr)
     (let ([fval (eval fun-expr sc)])
       (cases fval
         [(Fun bound-id bound-body)
          (eval bound-body
                (extend bound-id (eval arg-expr sc) sc))]
         [else (error 'eval "`call' expects a function, got: ~s"
                      fval)]))]))

(: run : String -> Number)
;; evaluate a FLANG program contained in a string
(define (run str)
  (let ([result (eval (parse str) empty-subst)])
    (cases result
      [(Num n) n]
      [else (error 'run
                   "evaluation returned a non-number: ~s" result)])))
```