מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: א
תאריך: 5/7/2022
שמות המרצים: מני אדלר, מיכאל אלחדד, ירון גונן
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה, שנה ב', סמסטר ב'
משך המבחן: 3 שעות
חומר עזר: אסור

הנחיות כלליות:

- יש לענות על כל השאלות בגיליון התשובות. מומלץ לא לחרוג מן המקום המוקצה.
- אם אינכם יודעים את התשובה, ניתן לכתוב 'לא יודע' ולקבל 20% מהניקוד על הסעיף/השאלה.

שאלה 1:	תחביר וסמנטיקה _____	35 נק
שאלה 2:	מערכת טיפוסים _____	20 נק
שאלה 3:	תכנות פונקציונאלי, CPS, רשימות עצלות	35 נק
שאלה 4:	תכנות לוגי _____	20 נק
סה"כ	_____	110 נק

בהצלחה!

שאלה 1: תחביר וסמנטיקה [35 נקודות]

א. הוספת הפרימיטיבים error, make-error? לשפה

בתרגיל 2 הגדרנו פונקציות משתמש לשם תמיכה בשגיאות.
כעת הוחלט להגדיר מנגנון זה, באופן מצומצם יותר, במסגרת השפה: האופרטורים הפרימיטיביים error?, make-error

```
(define div
  (lambda (x y)
    (if (= y 0)
      (make-error "div by 0")
      (/ x y))))
(define div1 (div 4 2))
(define div2 (div 4 0))
(error? div1)
→ false
(error? div2)
→ true
div1
→ 2
div2
→ <error: "div by 0">
```

א.1 הגדירו את הממשק Error המייצג ערך חדש בשפה L3:

```
type Value = SExpValue;
type SExpValue = number | boolean | string | PrimOp | Closure |
SymbolSExp | EmptySExp | CompoundSExp | Error;

export interface Error {

   ____________

   ____________
}

export const makeError = (_______________________________): Error =>

    ____________________________________________________;

export const isError =
_______________________________________________;
```

[6 נקודות]

א.2 הרחיבו את הפונקציה applyPrimitive כך שתתמוך בשתי הפעולות החדשות:

```
export const applyPrimitive = (proc: PrimOp, args: Value[]):
Result<Value> =>
    proc.op === "+" ? (allT(isNumber, args) ?
          makeOk(reduce((x, y) => x + y, 0, args)) :
          makeFailure("+ expects numbers only")) :
    proc.op === "-" ? minusPrim(args) :
    proc.op === "*" ? (allT(isNumber, args) ?
          makeOk(reduce((x, y) => x * y, 1, args)) :
          makeFailure("* expects numbers only")) :
    proc.op === "/" ? divPrim(args) :
    proc.op === ">" ? makeOk(args[0] > args[1]) :
    proc.op === "<" ? makeOk(args[0] < args[1]) :
    proc.op === "=" ? makeOk(args[0] === args[1]) :
    proc.op === "not" ? makeOk(!args[0]) :
    proc.op === "and" ? isBoolean(args[0]) && isBoolean(args[1]) ?
          makeOk(args[0] && args[1]) :
          makeFailure('Arguments to "and" not booleans') :
    proc.op === "or" ? isBoolean(args[0]) && isBoolean(args[1]) ?
          makeOk(args[0] || args[1]) :
          makeFailure('Arguments to "or" not booleans') :
    proc.op === "eq?" ? makeOk(eqPrim(args)) :
    proc.op === "string=?" ? makeOk(args[0] === args[1]) :
    proc.op === "number?" ? makeOk(typeof (args[0]) === 'number') :
    proc.op === "boolean?" ? makeOk(typeof (args[0]) === 'boolean') :
    proc.op === "string?" ? makeOk(isString(args[0])) :

    ____________________________________________________________________
:

    ____________________________________________________________________
:
    makeFailure("Bad primitive op " + proc.op);
```

[4 נקודות]

א.3 סטודנטים בקורס הציעו לממש את make-error ו error? כצורות מיוחדות. האם הייתם מקבלים את הצעתם? נמקו בקצרה.
[3 נקודות]

_____

_____

_____

א.4 כדי להתמודד עם אפשרות של הפעלת פונקציה עם ארגומנט שערכו הוא Error במסגרת האינטרפרטר, עדכנו את הפונקציה L3applyProcedure כך שבמידה ואחד הפרמטרים או האופרטור הם Error, היא מחזירה שגיאה זו.

```
(+ 3 4)
→ 7
(+ (div 2 0) 4)
→  <Error: "div by 0">
(square 4)
→ 16
(square (div 2 0))
→ <Error: "div by 0">

const L3applyProcedure = (proc: Value, args: Value[], env:
Env): Result<Value> =>
```

_____

_____

```
isPrimOp(proc) ? applyPrimitive(proc, args) :
isClosure(proc) ? applyClosure(proc, args, env) :
makeFailure("Bad procedure " + JSON.stringify(proc));
```

[4 נקודות]

ב. מימוש normal order במודל הסביבות

ב.1 הראו דוגמת קוד שחישובה מסתיים ושעבורה applicative order יעיל יותר מ-normal order

_____

_____

_____

[2 נקודות]

ב.2 הראו דוגמת קוד שחישובה מסתיים ושעבורה normal order יעיל יותר מ-applicative order

_____

_____

[2 נקודות]
להלן שתי פונקציות מהאינטרפרטר של L4:

```
const eval = (exp: CExp, env: Env): Result<Value> =>
    isNumExp(exp) ? makeOk(exp.val) :
    isBoolExp(exp) ? makeOk(exp.val) :
    isStrExp(exp) ? makeOk(exp.val) :
    isPrimOp(exp) ? makeOk(exp) :
    isVarRef(exp) ? applyEnv(env, exp.var) :
    isLitExp(exp) ? makeOk(exp.val) :
    isIfExp(exp) ? evalIf(exp, env) :
    isProcExp(exp) ? evalProc(exp, env) :
    isLetExp(exp) ? evalLet(exp, env) :
    isLetrecExp(exp) ? evalLetrec(exp, env) :
    isAppExp(exp) ?
       bind(eval(exp.rator, env),
            (proc: Value) =>
               bind(mapResult((rand: CExp) =>
                eval(rand, env), exp.rands),
                (args: Value[]) =>
                   applyProcedure(proc, args))) :
    exp;

const applyProcedure = (proc: Value, args: Value[]): Result<Value> =>
    isPrimOp(proc) ? applyPrimitive(proc, args) :
    isClosure(proc) ? applyClosure(proc, args) :
    makeFailure(`Bad procedure ${JSON.stringify(proc)}`);

const applyClosure = (proc: Closure, args: Value[]): Result<Value>
=>{
     const vars = map((v: VarDecl) => v.var, proc.params);
     return evalSequence(proc.body, makeExtEnv(vars, args, proc.env));
}

export const applyPrimitive = (proc: PrimOp, args: Value[]):
Result<Value> =>
// the implementation the function is not relevant for this question
…
```

ב.3 האם קוד זה מממש את מודל ההצבה או את מודל הסביבות? נמקו בקצרה [2 נקודות]

ב.4 האם קוד זה מממש את applicative order או את normal order? נמקו בקצרה [2 נקודות]

ב.5 כדי לממש normal order במודל הסביבות:

- עדכנו את הטיפוס של השדה vals בממשק ExtEnv ובחתימת הפונקציות makeExtEnv בהתאם, ושנו בעקבות כך את applyEnv כך שחישוב המשתנים יהיה ב normal order:

```
export interface ExtEnv {
    tag: "ExtEnv";
    vars: string[];

    vals: ~~Value~~ _________ [];
    nextEnv: Env;
}

export const makeExtEnv = (vs: string[], vals: ~~Value~~______[], env:
Env): ExtEnv =>
    ({tag: "ExtEnv", vars: vs, vals: vals, nextEnv: env});

const applyExtEnv = (env: ExtEnv, v: string): Result<Value> =>
    env.vars.includes(v) ?
       makeOk(~~env.vals[env.vars.indexOf(v)]~~) :

       ____________________________________________________ :
       applyEnv(env.nextEnv, v);
```

- עדכנו את קוד הפונקציה eval, ואת חתימות הפונקציות applyProcedure, applyClosure, applyPrimitive (שהופיעו למעלה) בהתאם.

[10 נקודות]

שאלה 2: טיפוסים [20 נקודות]

בתרגיל 4, כזכור, נוספה לשפה L51 אפשרות של user-defined types, לפי ההגדרות הבאות:

```
<exp>  ::= <cexp> | <defineExp> | <defineTypeExp>
<cexp> ::= <atomicExp> | <procExp> | <litExp> | <ifExp> | <appExp> | <typecaseExp>

<defineTypeExp> ::= ( define-type <id> [( <id> <VarDecl>* )]* )
    / DefTypeExp(typeName:string, records:Record[])
    / Record(typeName:string, fields:VarDecl[])

<typecaseExp> ::= ( type-case <id> <CExp> ( <case-exp> )+  )
    / TypeCaseExp(typeName: string, val: CExp, cases: CaseExp[])

<case-exp> ::= (id (<varDecl>*) <cexp>+)
    / CaseExp(typeName: string, varDecls: VarDecl[], body: CExp[])
```

בשפה L51 נתן להגדיר טייפים חדשים בהתאם ל-disjoint union pattern:

```
(define-type Shape
    (circle (radius : number))
    (rectangle (width : number) (height : number)))

(define (area : (Shape -> number))
    (lambda ((s : Shape)) : number
        (type-case Shape s
            (circle (r) (* (* r r) 3.14))
            (rectangle (w h) (* w h)))))

(area (make-circle 1))
```

בשפה L51 קיימים יחסים של type/subtype בין ה-record types (לדוגמה circle ו-rectangle) וה-user defined types (בדוגמה Shape).

א. [10 נק'] בהינתן ההגדרות לעיל - עבור כל typing statement רשום האם ה-statement נכון - אם לא, הסבר למה:

- **{f:[Number->T1]} ⊢ (f 12): T1**

- **{x:circle, f:[Shape->T1]} ⊢ (f x):T1**

- **{x:Shape} ⊢ x: rectangle**

_____

- **{x:circle} ⊢ x:Shape**

_____

- **{f:[T1->Shape], g:[circle->T2], x:T1} ⊢ (g (f x)):T2**

_____

ב. [5 נק'] בהשוואה בין מערכת הטייפים של Java ו-TypeScript, מבחינים בין:

structural subtyping (as in TypeScript)
nominal subtyping (as in Java)

התבוננו בשפה L51. האם L51 תומכת ב-**structural** או ב-subtyping **nominal**?
תנו דוגמה כדי לתמוך בתשובתכם.

_____

_____

_____

_____

_____

_____

ג. [5 נק'] האם ההגדרה הבאה של טייפ ב-L51 מגדירה טייפ חוקי - הסבירו:

```
(define-type T1
    (rec1 (f11 : number) (f12 : T2))
    (rec2 (f21 : string)))
(define-type T2
    (rec3 (f31 : T1))
    (rec4 (f41 : T2)))
```

שאלה 3: תכנות פונקציונאלי, CPS, רשימות עצלות [35 נקודות]

א. [5 נק'] הסבירו מהי רקורסית זנב

_____

_____

_____

ב. [5 נק'] הסבירו מהי אופטימיזציה של רקורסיית זנב

_____

_____

_____

ג. [5 נק'] הפונקציה remove-duplicates מקבלת רשימה lst, ומחזירה רשימה המכילה את כל האיברים מ-lst אבל ללא כפילות איברים. סדר האיברים ברשימת התוצאה הוא אותו הסדר כמו ב-lst. עבור איבר שמופיע יותר מפעם אחת, רק ההופעה הראשונה שלו נשמרת. דוגמאות:

```
(remove-duplicates '(1 1 1 1 1)) ⇒ '(1)
(remove-duplicates '(a b b a)) ⇒ '(a b)
(remove-duplicates '(1 2)) ⇒ '(1 2)
```

השלימו את קוד הפונקציה (רמז: השתמשו בפונקציה `filter`)

```
;; [List<T> -> List<T>]
(define remove-duplicates
  (λ (lst)
    (if (empty? lst) empty
```

_____

_____

_____

_____

_____

ד. [10 נק'] כתבו את הפונקציה בסגנון CPS - השלימו את החתימה. השתמשו ב-`filter$$`.

(filter$$ pred$ lst cont)

```
;; Type: ____________________________________
(define remove-duplicates$
  (λ (lst cont)
    (if (empty? lst)
        (cont empty)

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________
```

ה. [10 נק] היזכרו בפונקציה reduce:

```
;; Type: [[T1 * T2 -> T2] * T2 * List(T1) -> T2]
;; Purpose: Combine all the values of s using reducer
;; Example: (reduce + 0 '(1 2 3)) => (+ 1 (+ 2 (+ 3 0)))
(define reduce
  (lambda (reducer initial s)
    (if (empty? s)
        initial
        (reducer (car s)
                 (reduce reducer initial (cdr s))))))
```

כתבו את הפונקציה remove-duplicates תוך שימוש ב-reduce, **ללא קריאה רקורסיבית**.

```
(define remove-duplicates
  (λ (l)
    (reduce
```

_____

_____

_____

_____

שאלה 4: תכנות לוגי [20 נקודות]

נתונים החוקים הלוגיים member, not_member
כזכור, X \= Y מצליח כאשר משתנה X אינו unifiable עם משתנה Y.

```
member(X, [X|_]).
member(X, [_|Ys]) :- member(X, Ys).

not_member(_, []).
not_member(X, [Y|Ys]):- X \= Y, not_member(X,Ys).
```

בשאלה זו נייצג קבוצה על ידי רשימה. כזכור, בקבוצה כל איבר מופיע פעם אחת.

א. ממשו את החוקים הלוגיים הבאים עבור קבוצות.
בחוקים intersection, union, disjoint, סדר האיברים בקבוצה השלישית נקבע על פי סדרם בשתי הקבוצות הראשונות.

אין להוסיף פרוצדורות עזר נוספות

```
% Signature: is_set(S)/1
% Purpose: check whether S is a set.
% ?- is_set([])
% true
% ?- is_set([1,2,3])
% true
% ?- is_set([1,2,1,3])
% false
```

```
% Signature: intersection(S1,S2,S3)/3
% Purpose: S3 is the intersection (חיתוך) of S1 and S2.
% ?-intersection([1,2,4],[2,3,1],[1,2])
% true
% ?-intersection([1,2],[3,4],[])
% true
% ?-intersection([1,1],[1],[1])
% false
```

```
% Signature: union(S1,S2,S3)/3
% Purpose: S3 is the union (איחוד) of S1 and S2.
% ?-union([1,2],[3],[1,2,3])
% true
% ?-union([1,2],[3,3],[1,2,3])
% false
```

```
% Signature: difference(S1,S2,S3)/3
% Purpose: S3 is the difference between S1 and S2 (S1 - S2) .
% ?- difference([1,2,3],[1],[2,3])
% true
% ?- difference([1,2],[3,4],[1,2])
% true
% ?-difference([1],[1],[])
% true
% ?-difference([1,1,2],[2],[1])
% false
```

[15 נקודות]

ב. תנו דוגמאות לשאילתות על קבוצות (החוקים מסעיף א) שעץ ההוכחה שלהן הוא:

- עץ הצלחה סופי

_____-?

- עץ כישלון סופי

_____-?

- עץ הצלחה עם אינסוף תשובות

_____-?

[דוגמא אחת לכל סוג של עץ. 5 נקודות]