מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
תאריך: 1/7/2019
שמות המרצים: מני אדלר, בן אייל, גיל אינציגר, מיכאל אלחדד, ירון גונן
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה, שנה ב', סמסטר ב'
משך המבחן: 3 שעות
חומר עזר: אסור

הנחיות כלליות:

- מבחנים שיכתבו בעיפרון חלש, המקשה על הקריאה, לא יבדקו.
- 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 _____	25 נק
שאלה 3:	CPS, Generators ורשימות עצלות __	30 נק
שאלה 4:	תכנות לוגי _____	20 נק
סה"כ	_____	110 נק

בהצלחה!

שאלה 1: תחביר וסמנטיקה [35 נקודות]

בשאלה זו, נרחיב את השפה L2 כך שתכלול מבנה של לולאות.

הצורה המיוחדת for כוללת את משתנה הלולאה, את הערך ההתחלתי ואת ערך הסיום של משתנה זה בלולאה, ואת הביטוי לחישוב. בכל לולאה מחושב הביטוי על פי ערכו הנוכחי של משתנה הלולאה. ערך ביטוי ה-for הוא ערך הביטוי בלולאה האחרונה.

לדוגמא: בחישוב הביטוי `(for n 1 3 (* n n))`, יחושב התת-ביטוי (n n *) שלוש פעמים: עבור n=1, עבור n=2, ועבור n=3.
הערך של כל הביטוי הוא החישוב (n n *) עבור n=3 (הלולאה האחרונה) - 9.

שימו לב, כי הביטוי של הערך ההתחלתי והערך הסופי בלולאה (1 3 בדוגמא שהובאה קודם) יכול להיות ביטוי מספרי אך גם ביטוי שערכו מספרי (כמו x y, כאשר הם הוגדרו קודם עם ערך מספרי) - למשל:

```
(for n x (* 2 x) (+ n x))
```

1.1 [4 נקודות]

הרחיבו את התחביר של L2 (קונקרטי ואבסטרקטי) כך שיכלול את הצורה המיוחדת החדשה for.

```
<program> ::= (L2 <exp>+) // program(exps:List(exp))
<exp> ::= <define-exp> | <cexp>
<define-exp> ::= (define <var-decl> <cexp>) // def-exp(var:var-decl,
val:cexp)
<cexp> ::= <num-exp> // num-exp(val:Number)
       | <bool-exp>  // bool-exp(val:Boolean)
       | <prim-op>   // prim-op(op:string)
       | <var-ref>   // var-ref(var:string)
       | (if <exp> <exp> <exp>) // if-exp(test,then,else)
       | (lambda (<var-decl>*) <cexp>+)
                        // proc-exp(params:List(var-decl), body:List(cexp))
       | (<cexp> <cexp>*) // app-exp(rator:cexp, rands:List(cexp))
<prim-op> ::= + | - | * | / | < | > | = | not
<num-exp> ::= a number token
<bool-exp> ::= #t | #f
<var-ref> ::= an identifier token
<var-decl> ::= an identifier token
```

1.2 [6 נקודות]

השלימו (**בגיליון התשובות**) את מימוש ה ADT של המבנה החדש:

```
interface ForExp {...};
const makeForExp = (...): ForExp => ...;
const isForExp = (x: any): ... => ...;
```

1.3 [10 נקודות]

השלימו את פרוצדורת ה eval באינטרפרטר של L2 במימוש הפרוצדורה **evalForExp** עבור הצורה המיוחדת for, ע"פ מודל הסביבות (ניתן להשתמש בלולאת for של TypeScript):

```
const L2eval = (exp: CExp | Error, env: Env): Value | Error =>
   isError(exp) ? exp :
   isNumExp(exp) ? exp.val :
   isBoolExp(exp) ? exp.val :
   isPrimOp(exp) ? exp :
   isVarRef(exp) ? applyEnv(env, exp.var) :
   isForExp(exp) ? evalForExp(exp, env) :
   isProcExp(exp) ? makeClosure(exp.args, exp.body) :
   isAppExp(exp) ? applyProc(L2eval(exp.rator, env),
                        map((r) => L2eval(r, env), exp.rands), env) :
   Error(`Bad L2 AST ${exp}`)

const evalForExp = (exp: ForExp, env: Env): Value | Error => {
    //@TODO
}
```

1.4 [10 נקודות]

1.4.1 המירו את ביטוי ה-for הבא לשני ביטויים של AppExp (עם אופרטור מסוג פרוצדורה, לא אופרטור פרימיטיבי) שמייצגים את אתו חישוב שה-for מייצג:

```
(for n 1 2 (* n n)) → ...
```

1.4.2 ממשו את הפרוצדורה For2Apps הממירה ביטוי מסוג ForExp למערך שקול של AppExp:

```
const For2Apps = (exp : ForExp, env: Env) : AppExp[] =>
    //@TODO
```

1.5 [5 נקודות]

ציירו את דיאגרמת הסביבות עבור חישוב התוכנית כאשר חישוב ה-for הוא לפי החלפה ברצף של הפעלות פרוצדורות (בעזרת For2Apps):

```
(define x 5)
(for n 1 2 (+ x n))
```

שאלה 2: מערכת טיפוסים [25 נקודות]

חיתוך בין טיפוסים כתוב ב-TypeScript כ: **T1 & T2**
הטיפוס **T1 & T2** מייצג את קבוצת הערכים המוכלים גם בקבוצת הערכים ש-**T1** מייצג וגם בזו ש-**T2** מייצג.
נגדיר type constructor חדש בשם **range** המייצג תת-קבוצה (בדרך כלל אין-סופית) של הערכים מתוך טיפוס מסודר הנמצאים בין שני ערכים. למשל:

```
range(1,10)
```

מייצג את קבוצת המספרים מ-1 עד 10 (כולל).

```
range("ab", "dzz")
```

מייצג את קבוצת ה-strings מ-"ab" עד "dzz" (כולל).

2.1 [6 נק]

2.1.1

בהינתן ההגדרות:

```
interface I1 {a: number; b: string};
Interface I2 {a: number: c: boolean};
```

כתבו את הגדרת ה-type I3 שמייצג אותם ערכים כמו I1 & I2.

2.1.2

בהינתן ההגדרות:

```
type R1 = range(1,10);
type R2 = range(2,12);
```

כתבו את הגדרת ה-type שמייצג אותם ערכים כמו R1 & R2.

2.1.3

בהינתן הגדרות ה-functional types:

```
type F1 = (a: number) => boolean;
type F2 = (a: string) => boolean;
```

כתבו את הגדרת ה-functional type שמייצג אותם ערכים כמו F1 & F2. זכרו שפרמטרים הם contra-variant ב-type checking.

2.2 [8 נק]

נרחיב את הגדרת TExp לשפה L5 כדי לתמוך ב-type intersection ו-range types.

```
<texp>          ::= <atomic-te> | <composite-te> | <tvar>
<atomic-te>     ::= <num-te> | <bool-te> | <void-te>
<num-te>        ::= number   // num-te()
<bool-te>       ::= boolean  // bool-te()
<str-te>        ::= string   // str-te()
<void-te>       ::= void     // void-te()
<composite-te> ::= <proc-te> | <tuple-te>
<non-tuple-te> ::= <atomic-te> | <proc-te> | <tvar>
<proc-te>       ::= [ <tuple-te> -> <non-tuple-te> ]
                              // proc-te(param-tes: list(te), return-te: te)
<tuple-te>      ::= <non-empty-tuple-te> | <empty-te>
<non-empty-tuple-te> ::= ( <non-tuple-te> *)* <non-tuple-te>
                              // tuple-te(tes: list(te))
<empty-te>      ::= Empty
<tvar>          ::= a symbol starting with T  // tvar(id: Symbol)
<inter-te>      ::= [ <texp> & <texp> (& <texp>)* ]
                              // inter-te(param-tes: list(te))
<range-te>      ::= [ range <litExp> <litExp> ]
                              // range-te(low: litExp, high: litExp)
```

2.2.1 הרחיבו את הגדרת ה-types ב-TExp.ts בהתאם ל-BNF הנזכר מעלה:

```
type TExp =  AtomicTExp | CompoundTExp | TVar;
const isTExp = (x: any): x is TExp =>
      isAtomicTExp(x) || isCompoundTExp(x) || isTVar(x);
type AtomicTExp = NumTExp | BoolTExp | StrTExp | VoidTExp;
const isAtomicTExp = (x: any): x is AtomicTExp =>
      isNumTExp(x) || isBoolTExp(x) || isStrTExp(x) || isVoidTExp(x);

type CompoundTExp = ProcTExp | TupleTExp;
const isCompoundTExp = (x: any): x is CompoundTExp =>
      isProcTExp(x) || isTupleTExp(x);

type NonTupleTExp = AtomicTExp | ProcTExp | TVar;
const isNonTupleTExp = (x: any): x is NonTupleTExp =>
      isAtomicTExp(x) || isProcTExp(x) || isTVar(x);


// proc-te(param-tes: list(te), return-te: te)
type ProcTExp = { tag: "ProcTExp"; paramTEs: TExp[]; returnTE: TExp; };
const makeProcTExp = (paramTEs: TExp[], returnTE: TExp): ProcTExp =>
      ({tag: "ProcTExp", paramTEs: paramTEs, returnTE: returnTE});
```

```
const isProcTExp = (x: any): x is ProcTExp => x.tag === "ProcTExp";

type TupleTExp = NonEmptyTupleTExp | EmptyTupleTExp;
const isTupleTExp = (x: any): x is TupleTExp =>
      isNonEmptyTupleTExp(x) || isEmptyTupleTExp(x);

interface EmptyTupleTExp { tag: "EmptyTupleTExp" };
const makeEmptyTupleTExp = (): EmptyTupleTExp =>
      ({tag: "EmptyTupleTExp"});
const isEmptyTupleTExp = (x: any): x is EmptyTupleTExp =>
      x.tag === "EmptyTupleTExp";

// NonEmptyTupleTExp(TEs: NonTupleTExp[])
interface NonEmptyTupleTExp {tag: "NonEmptyTupleTExp"; TEs: NonTupleTExp[]; };
const makeNonEmptyTupleTExp = (tes: NonTupleTExp[]): NonEmptyTupleTExp =>
      ({tag: "NonEmptyTupleTExp", TEs: tes});
const isNonEmptyTupleTExp = (x: any): x is NonEmptyTupleTExp =>
      x.tag === "NonEmptyTupleTExp";
```

כתבו רק השורות החדשות או את אלה שצריך לשנות כדי לתמוך ב-range-te ו-inter-te:
ענו בגיליון התשובות.

2.2.2

בהינתן שפעולת החיתוך היא commutative ו-associative ושעבור כל טיפוס T&T=T, מחליטים שהייצוג הפנימי של ביטוים מסוג inter-te יהיה רשימה שטוחה וממוינת של ה-components של ה-inter-te (כפי שנעשה ב-union-te בתרגיל 4). ה-components ממוינים בסדר אלפביתי לפי צורת unparse של ה-component.

כתבו את הערך הפנימי של ה-AST מסוג InterTExp עבור ביטוי ה-types הבאים:

ענו בגיליון התשובות.

```
(number & boolean) →

{tag: "InterTExp", ….}

(number & (string & boolean)) →

...

(number & (boolean & number)) →

...

((range 1 10) & number) →

...
```

2.3 [5 נק.]

כתבו את החוקים הקובעים ששני types הם מתאימים (compatible) כשמתחשבים ב-type intersection: (**השלימו בגיליון התשובות**)

We say that T1 and T2 are compatible if T1 can be used in the places where T2 is expected. In other words, T1 is compatible with T2 when T1 is a subtype of T2.

1. AtomicType1 and AtomicType2 are compatible only if they are identical
2. AtomicType1 and (T1 & T2) are compatible if: ……
3. AtomicType1 and ProcExpT2 are not compatible.

4. (T1 & T2) and AtomicType2 are compatible if: …..
5. (T11 & T12) and (T21 & T22) are compatible if: ….
6. (T1 & T2) and ProcTExp2 are compatible if <no need to specify>
7. ProcTExp1 and AtomicType2 are not compatible.
8. ProcTExp1 and (T1 & T2) are compatible if <no need to specify>
9. ProcTExp1 and ProcTExp2 are compatible <unchanged - no need to specify>.
10. Any type is compatible with TVar

2.4 [6 נק.]

השלם את התוכנית L5 הבאה כך שאחד מהתת-ביטויים יקבל type intersection כ-type בסוף תהליך type inference:

```
(L5 (define f (lambda ((x : (range 1 10)) : boolean #t)))

    (define g (lambda ((y : _________________)) : number 0))

    (define (p : ________________________) ___________________)

    (and (f ________________________)

         (> (g _______________________) 0)))
```

שאלה 3: CPS, Generators ורשימות עצלות [30 נקודות]

3.1 [4 נקודות]

מה ההבדל בין תהליך חישוב רקורסיבי לתהליך חישוב איטרטיבי?

3.2 [4 נקודות]

תנו תרחיש שימוש (use case) בו עדיף שימוש ברשימה עצלה על פני רשימה רגילה, והסבירו מדוע הוא עדיף (להסביר במילים, אין צורך בקוד)

3.3 [10 נקודות]

השלימו את הגדרת הפונקציה `take-while$`, המקבלת פרדיקט, רשימה עצלה ו-continuation, ומחשבת רשימה (רגילה) של כל האיברים מתוך הרשימה העצלה לפי הסדר, עד האיבר הראשון שאינו מקיים את הפרדיקט. השתמשו בפונקציות הממשק עבור רשימות עצלות: `empty-lzl`, `head` ו-`tail`.

דוגמה: (`ints` היא רשימת כל המספרים הטבעיים ו-`identity` היא פונקצית הזהות)

```
(take-while$ (lambda (n) (< n 5)) ints identity) ;; => '(0 1 2 3 4)
```

3.4 [10 נקודות]: Generators ב-JavaScript

השלימו את הגדרת הפונקציה `takeWhile` המקבלת generator ופרדיקט ומחזירה מערך של איברים מה-generator כל עוד האיברים מקיימים את הפרדיקט, בדומה לסעיף הקודם.

לדוגמה, בהינתן ה-generator הבא:

```
function* numbers() {
   yield 1;
   yield 3;
   yield 5;
   yield 6;
   yield 7;
   return 8;
}
```

מתקיים:

```
takeWhile(numbers(), x => x % 2 === 1); // => [1, 3, 5]
```

3.5 [2 נקודות]

מה יקרה אם הקלט לפונקציות בסעיפים 3.3 ו-3.4 יהיה רשימות עצלות/generators אינסופיים כאשר כל האיברים מקיימים את הפרדיקט?

שאלה 4: תכנות לוגי [20 נקודות]

מטרת שאלה זו היא לממש את המספרים הטבעיים באמצעות מנגנון הרשימות של פרולוג. המספר הטבעי המיוצג על ידי רשימה (סופית) הוא אורך הרשימה. לדוגמא: הרשימה [] מייצגת את המספר 0 וכן הרשימות [1,2,3] ו [1,1,1] מייצגות את המספר 3.

סעיף 4.1 [5 נקודות]

ממשו את הפרדיקט natural\1 המתקיים כאשר הפרמטר הוא מספר טבעי בייצוג רשימות.

```
% Signature: natural(X)/1
% Purpose: X is a (list representation) of a number.
```

סעיף 4.2 [5 נקודות]

ממשו את היחס smaller\2 שמקבל שני מספרים טבעיים ומכריע האם המספר השמאלי קטן ממש מהימני. לדוגמא:

```
smaller([],[]) – False.
smaller([],[1]) – True.
smaller([1,2],[1]) – False

% Signature: smaller(X, Y)/2
% Purpose: X<Y on list representations.
```

סעיף 4.3 [5 נקודות]

ממשו את פעולת החיבור על ייצוג מספרים כרשימות:

```
% Signature: plus(X, Y, Z)/3
% Purpose: Z is the sum of X and Y
% X, Y and Z are numbers represented as lists.
```

סעיף 4.4 [5 נקודות]

ממשו את פעולת הכפל:

```
% Signature: times(X,Y,Z)/3
% Purpose: Z = X*Y
% X, Y and Z are numbers represented as lists.
```