תאריך 23.07.2018
שם המרצים: מני אדלר, מיכאל אלחדד, ירון גונן
מבחן בקורס: עקרונות שפות תכנות
קורס' מס: 202-1-2051
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה
שנה: ב' סמסטר: ב' מועד ב'
משך הבוחן: 3 שעות
חומר עזר: אסור

הנחיות כלליות:

1. **ההוראות במבחן מנוסחות בלשון זכר, אך מכוונות לנבחנים ולנבחנות כאחד.**
2. **מבחן הכתוב בעיפרון חלש המקשה על הקריאה, לא יבדק**
3. **יש לענות על כל השאלות בגוף המבחן בלבד (בתוך השאלון). מומלץ לא לחרוג מהמקום המוקצה.**
4. **אם אינך יודע את התשובה, ניתן לכתוב "לא יודע" ולקבל 20% מהניקוד על הסעיף/השאלה.**
5. **18 דפים סה"כ במבחן**

שאלה 1: AST	**נק 26**
שאלה 2: ייצוג אובייקטים ב-L	**נק 14**
שאלה 3: מערכת טיפוסים	**נק 23**
שאלה 4: תכנות לוגי	**נק 21**
שאלה 5: רשימות עצלות וCPS	**נק 22**
סה"כ	**נק 106**

שאלה 1: AST

(26 נק)

בשאלה הזאת נפתח שיטה ליצירה של קוד ב-L5 בהינתן תיאור של Type.

1.1 התבונן ב-Type הבא ב-TypeScript:

(2 נק)

```
interface Student {tag: "Student", name: string; age: number;}
```

הגדרו את ה-value constructor ואת ה-type predicate עבור ה-type Student:

```
const makeStudent = (_________________________________) : _________ =>

_____________________________________________________________

const isStudent = (_______________) : _________________ =>

_____________________________________________________________
```

1.2 נעביר את הקוד ל-L5:

כדי לממש את ה-Type Student ב-L5, הגדרו את הפרוצדורות הבאות:

(2 נק)

```
(define makeStudent (lambda ((name : string) (age : number)) : list
   (list "Student" name age)))

(define student? (lambda (_________) : ___________

_____________________________________________)

(define student->name ___________________________________

_____________________________________________)
```

1.3 ציירו את ה AST של הביטוים L5 של define עבור makeStudent, ושל ביטוי ה define עבור student?
(8 נק)

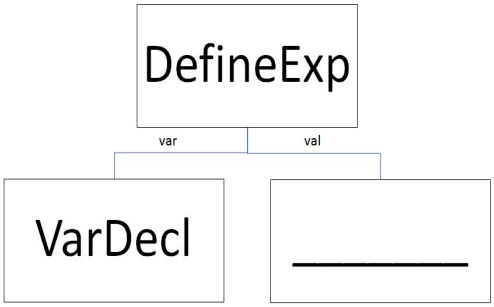
TEXP

```
<texp>          ::= <atomic-te> | <compound-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()
<list-te>       ::= list // list-te()
<compound-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))
<tvar>          ::= a symbol starting with T
                  / tvar(id: Symbol, contents, Box(string|boolean))
```

L5

```
<exp> ::= <define> | <cexp>                / DefExp | CExp
<define> ::= ( define <var-decl> <cexp> )
               / DefExp(var:VarDecl, val:CExp)
<cexp> ::= <number>                        / NumExp(val:number)
        |  <boolean>                       / BoolExp(val:boolean)
        |  <string>                        / StrExp(val:string)
        |  <var-ref>
        |  ( lambda ( <var-decl>* ) <TExp>* <cexp>+ )
            / ProcExp(params:VarDecl[], body:CExp[], returnTE: TExp))
        |  ( if <cexp> <cexp> <cexp> )
            / IfExp(test: CExp, then: CExp, alt: CExp)
        |  ( <cexp> <cexp>* )
            / AppExp(operator:CExp, operands:CExp[]))
<prim-op>  ::= + | - | * | / | < | > | = | not |  eq? | string=?
                  | cons | car | cdr | list | list? | number?
                  | boolean? | symbol? | string?
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<var-ref>  ::= an identifier token                / VarRef(var: string)
<var-decl> ::= an identifier token | (var : TExp) / VarRef(var, TE: TExp)
```

makeStudent



student?

1.4 השלימו את שני הביטויים הבאים, האחד בונה את ה-AST שמייצג את ה-value constructor makeStudent, והשני בונה את ה-AST שמייצג את ה type predicate student?

(6 נק)

```
const studentCtorExp : ________________________ =
   makeDefineExp(
     makeVarDecl("makeStudent", makeFreshTVar()),
     makeProcExp([makeVarDecl("name", makeStrTExp()),
                  makeVarDecl("age",  makeNumTExp())],
                 [makeAppExp(makePrimOp("list"),
                       [________________________,

                        ________________________,

                        ________________________])],
                 makeListTExp()));


const studentPredExp : ____________________________________ =
   makeDefineExp(
     makeVarDecl("student?", makeFreshTVar()),
     makeProcExp(

       ____________________________________________________,
       [makeAppExp(makeVarRef("string=?"),

                   [makeAppExp(____________________________________

                               ________________________________________),
                   makeStrExp("Student")])]),
       makeBoolTExp());
```

1.5 יצירת קוד

ברצוננו ליצור את הקוד של ה-value constructor וה-type predicate בצורה אוטומטית בהינתן תיאור הצהרתי (declarative) של ה-type.

(8 נק)

```
interface Field {name: string; typeName: string;}
interface Record {name: string; fields: Field[];}

const studentRecord : Record = {name: "Student",
          fields: [{name: "name", typeName: "string"},
                   {name: "age", typeName: "number"}]};

const studentCtorAuto = genCtor(studentRecord);
Const studentPredAuto = genPred(studentRecord);

console.log(unparse(studentCtorAuto))
→ (define makeStudent (lambda ((name : string) (age : number)) : list
(list "Student" name age)))
```

השלמו את הקוד של ה-code generator:

```
const genCtor = (rec: Record): ______________________________ =>
 makeDefineExp(
   makeVarDecl("make"+rec.name, makeFreshTVar()),
   makeProcExp(
     map((field : Field) => ____________________________________________),
         rec.fields),
      [makeAppExp(makeVarRef("list"),
        [makeStrExp(rec.name)].
          concat(map((field : Field) => ________________________________,
                     rec.fields)))],
     makeListTExp()));
```

שאלה 2: ייצוג אובייקטים ב-L

(14 נקודות)

נתונים שלושה מימושים לייצוג אובייקט של בן-אדם (אחד ב TypeScript ושניים ב L5), וכן פונקציה הבודקת האם הוא גר בניו-יורק:

מימוש 1: כ-interface ב-TypeScript

```
interface Person {
    tag: "person";
      name: string;
    address: string
};

const isLiveInNY = (p : Person) : boolean =>
   p.address === "NY";
```

מימוש 2: כרשימה ב-L5

```
(define make-person-l
  (lambda ((name : string)
           (address : string)) : List
      (list "person" name address)))

(define is-live-in-NY-l
   (lambda ((p : List)) : boolean
      (and (eq? (car p) "person")
           (eq? (car (cdr (cdr p))) "NY"))))
```

מימוש 3: כ-closure ב-L5

```
(define make-person-c
  (lambda ((name : string)
           (address : string)) : (symbol -> T)
    (lambda (msg)
      (cond ((eq? msg 'tag) "person")
            ((eq? msg 'name) name)
            ((eq? msg 'address) address)))))
```

```
(define is-live-in-NY-c
   (lambda (p : (symbol -> T)) : boolean
     (and (eq? (p 'tag) "person")
          (eq? (p 'address) "NY"))))
```

2.1 ציינו את הערך של כל אחד משלושת הביטויים הבאים (כל אחד מהביטויים מתייחס, בהתאמה, לאחד משלושת המימושים למעלה):
(3 נקודות)

```
isLiveInNY({tag: "variable", name: "x", address: "NY"});
```

_____

```
(is-live-in-NY-l (list "variable" "x" "NY"))
```

_____

```
(is-live-in-NY-c
  (lambda (msg)
    (cond ((eq? msg 'tag) "variable")
          ((eq? msg 'name) "x")
          ((eq? msg 'address) "NY"))))
```

_____

2.2 מדוע אין צורך לבדוק את הערך של tag במימוש של **isLiveInNY**,אך צריך לבדוק את הערך של tag בשני המימושים האחרים **is-live-in-NY-c, is-live-in-NY-l** מה ההבדל העקרוני בתפקיד של tag בייצוג של Person ב TypeScript ובייצוגים שלו ב L5?

(5 נקודות)

_____

_____

_____

_____

2.3 נתונים שני אובייקטים p-c,p-l המייצגים בן אדם ע"פ שני המימושים ב L5,

כרשימה וכ-closure:

(6 נקודות)

```
(define p-l (make-person-l "Danny" "Beer Sheva"))
(define p-c (make-person-c "Danny" "Beer Sheva"))
```

עבור כל אחד מהמקרים הבאים, ציינו היכן מאוחסנים ה'שדות' של האובייקט (name, address):

האובייקט p-l, כאשר האינטרפרטר במודל הסביבות (environment model)

1. בסביבה הגלובלית
2. באחת הסביבות שאינה הגלובלית
3. כחלק מהקוד של body ב closure
4. בזיכרון של האינטרפרטר
5. במקום אחר

האובייקט p-c, כאשר האינטרפרטר במודל ההצבה (substitution model)

1. בסביבה הגלובלית
2. באחת הסביבות שאינה הגלובלית
3. כחלק מהקוד של body ב closure
4. בזיכרון של האינטרפרטר
5. במקום אחר

האובייקט p-c, כאשר האינטרפרטר במודל הסביבות (environment model)

1. בסביבה הגלובלית
2. באחת הסביבות שאינה הגלובלית
3. כחלק מהקוד של body ב closure
4. בזיכרון של האינטרפרטר
5. במקום אחר

שאלה 3: מערכת טיפוסים

(23 נקודות)

בשאלה זו נרחיב את מערכת הטיפוסים שהוגדרה ב L5 כדי לתמוך באיחוד של טיפוסים (union types) כמו ב TypeScript.

לדוגמא, ניתן יהיה להגדיר טיפוס חדש של קבוצת כל המחרוזות וכל המספרים כאיחוד של string ו number:

```
(string | number)
```

לשם פשטות נניח כי ניתן לבצע union רק של טיפוסים לא מורכבים.

3.1. מוטיבציה: תארו מקרה בו נדרש להגדיר טיפוס שהוא איחוד של טיפוסים (כמו פרוצדורה בה נדרש לאחד טיפוסים כדי להגדיר את אחד הפרמטרים או את הערך המוחזר).
(3 נקודות)

3.2. נתון התחביר המופשט והקונקרטי של מערכת הטיפוסים, כפי שנלמד בהרצאה. הרחיבו אותו עם מבנה של איחוד טיפוסים.
(3 נקודות)

```
<texp>          ::= <atomic-te> | <composite-te> | <tvar>
<atomic-te>     ::= <num-te> | <bool-te> | <void-te> | <str-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)
```

_____

_____

_____

3.3. השלימו את הגדרת מבנה ה union ב AST
(3 נקודות)

```
export type UnionTExp = { tag: "UnionTExp";

            ____________________________________________________ };

export const makeUnionTExp =

   (_________________________________________): UnionTExp =>

   ({tag: "UnionTExp";  ____________________________________});

export const isUnionTExp = (x: any): x is UnionTExp =>
    x.tag === "UnionTExp";
```

3.4 מתי ניתן לקבוע ש-type T1 יותר ספציפי מ-type T2?
(2 נק)

3.5. נניח כי מומש ב Parser הניתוח של מבנה איחוד הטיפוסים, כך שתאור טיפוס מאוחד, בקוד כמו ' number | string', יהפוך ב parsing לקודקוד מסוג UnionTExp ב AST (אופן המימוש אינו רלבנטי לשאלה זו).
(12 נקודות)

עדכנו את המתודה checkEqualType ב TypeChecker כך שייבדקו גם מקרים בהם הטיפוסים כוללים union types:

- במידה ואחד הפרמטרים הוא טיפוס מסוג union type, הטיפוס בפרמטר הראשון te1 צריך להיות **ספציפי יותר** מהטיפוס בפרמטר השני te2.
  לדוגמא: אם
  ```
  te1 = number, te2 = (number | string)
  ```
  אז הפרוצדורה מחזירה true

- אם
  ```
  te1 = boolean, te2 = (number | string)
  ```
  אז הפרצדורה מחזירה error

- במידה ושני הפרמטרים הם מטיפוס פרוצדורה, יש לוודא תאימות של טיפוס הפרמטרים והערך המוחזר גם עבור union types.
  לדוגמא:
  אם
  ```
  te1 = (number -> boolean)
  te2 = ((number | string) -> boolean)
  ```
  אז הפרוצדורה מחזירה true
  אם
  ```
  te1 = (number -> (boolean | string))
  te2 = (number -> boolean)
  ```
  אז הפרוצדורה מחזירה error

השתמשו בממשק ל-AST של TExp:

```
isUnionTExp, isProcTExp, isAtomicTExp, isTVar
And for values p of type procTExp - p.paramTEs and p.returnTE.

// Purpose: Check that type expressions are equivalent
// as part of a fully-annotated type check process of exp.
// Return an error if the types are different - true otherwise.
// Exp is only passed for documentation purposes.
const checkEqualType = (te1: TExp | Error,
                        te2: TExp | Error,
                        exp: Exp): true | Error =>
  isError(te1) ? te1 :
  isError(te2) ? te2 :
  deepEqual(te1, te2) ? true :
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

```
:
Error(`Incompatible types: ${unparseTExp(te1)} and
${unparseTExp(te2)} in ${unparse(exp)}`);
```

שאלה 4: תכנות לוגי

(21 נק)

4.1: הסבירו בקצרה מדוע כל תוכנית ב-Relational Logic Programming היא ברת הכרעה:
(3 נק)

היזכרו בהגדרת מספרי צ'רץ':

```
natural_number(0).
natural_number(s(N)) :- natural_number(N).
```

נגדיר את היחס `gt` אשר מתקיים כאשר הארגומנט הראשון גדול יותר מן הארגומנט השני:

```
gt(s(X),0) :- natural_number(X).
gt(s(X), s(Y)) :- gt(X,Y).
```

4.2: פרטו את צעדי החישוב עבור מציאת ה-unifier הבא:
(3 נק)

```
unify( gt(s(s(0)), X), gt(s(X), 0) )
```

4.3 הוסיפו את היחס lte/2 אשר מתקיים כאשר X קטן או שווה ל-Y
(5 נק)

```
lte( 0, 0 ).

lte( ____________________) :- ________________________________

lte(_____________________) :- ________________________________
```

4.4: השלימו את הקוד הבא עבור פרדיקט אשר בודק האם איבר מסוים נמצא במקום הנכון ברשימה ממוינת (של מספרי צ'רץ'):
(5 נק)

```
insert(H, [], [H]).insert(H, [X|Y], [H,X|Y]) :- gt(X, H).
insert(H, [X|Y], [X|Z]) :- insert(H, Y, Z).


insert(H, [], [H]).

insert(H, [X | Y], [H, X | Y]) :- ______________________________.

insert(H, [X|Y], [X|Z]) :- ____________________________________.
```

4.5: השלימו את הקוד הבא אשר ממין רשימה:
(5 נק)

```
sort([],[]).

sort([H | T], R) :-

_____________________________________________________

_____________________________________________________

_____________________________________________________
```

שאלה 5: רשימות עצלות ו-CPS

(22 נק)

5.1: האם ניתן להשוות בין שתי רשימות עצלות? אם כן, כתבו את הפרדיקט `?lzl-equal` אם לא, הסבירו מדוע. **(3 נק)**

_____

_____

_____

_____

5.2: ניזכר באלגוריתם של ניוטון לחישוב שורש של מספר: **(9 נק)**

מטרה: חשב $\sqrt{x}$:

1. המשתנה y הוא ניחוש של הערך $\sqrt{x}$ (נתחיל תמיד עם ניחוש של 1)
2. ניחוש מוצלח יותר מתקבל מהחישוב הבא: $\frac{y+\frac{x}{y}}{2}$
3. חזור על החישוב עד שהניחוש בריבוע שווה בערך ל-x

כתבו את הרשימה העצלה sqrt-lzl אשר מכילה את כל הניחושים של השורש מהאלגוריתם. השתמשו בממשקים לעבודה עם רשימות עצלות: cons-lzl, head, tail. כמו כן נתונה לכם הפונקציה המחשבת את הניחוש המוצלח יותר, improve.

דוגמה:

```
> (take (sqrt-lzl 2) 5)
'(1 1.5 1.4166666666666665 1.4142156862745097 1.4142135623746899)
```

```
(define improve
  (λ (guess x)
    (/ (+ guess (/ x guess)) 2.0)))
```

```
(define sqrt-lzl
  (λ (x)
```

_____

_____

_____

_____

_____

_____

_____

_____

CPS 5.3

(10 נק)

ממשו את הפרוצדורה filter$$ שמקבלת פרדיקט pred$ שכתוב בצורת CPS רשימה ו-continuation cont- ומעבירה ל-cont את רשימת האיברים המקיימים את הפרדיקט.

```
;; Purpose: filter in CPS

;; Type: ____________________________________________________

(define filter$$
    (lambda (pred$ lst cont)

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________
```