מבחן בקורס: עקרונות שפות תכנות, 202-1-2051
מועד: א
תאריך: 22/7/2024
שמות המרצים: מני אדלר, ירון גונן, יובל פינטר
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה
משך המבחן: שעתיים
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	30 נק
שאלה 4:	תכנות לוגי _____	15 נק
סה"כ	_____	100 נק

בהצלחה!

שאלה 1: תחביר וסמנטיקה [35 נקודות]

בתרגיל 2 הוספנו כזכור מבנה של class לשפה L3:

```
<program> ::= (L31 <exp>+)              / Program(exps:List(exp))
<exp> ::= <define> | <cexp>             / DefExp | CExp
<define> ::= ( define <var> <cexp> )    / DefExp(var:VarDecl,
val:CExp)
<var> ::= <identifier>                  / VarRef(var:string)
<cexp> ::= <number>                     / NumExp(val:number)
        |  <boolean>                    / BoolExp(val:boolean)
        |  <string>                     / StrExp(val:string)
        |  ( lambda ( <var>* ) <cexp>+ ) / ProcExp(args:VarDecl[],
        |                                /         body:CExp[]))
        |  ( class ( <var>+ ) ( <binding>+ ) )
                 / ClassExp(fields:VarDecl[], methods:Binding[]))
        |  ( if <cexp> <cexp> <cexp> ) / IfExp(test: CExp,
        |                                      then: CExp,
        |                                      alt: CExp)
        |  ( let ( <binding>* ) <cexp>+ ) /
LetExp(bindings:Binding[],
        |                                      body:CExp[]))
        |  ( quote <sexp> )                / LitExp(val:SExp)
        |  ( <cexp> <cexp>* )              / AppExp(operator:CExp,
        |                                         operands:CExp[]))
<binding>  ::= ( <var> <cexp> )            / Binding(var:VarDecl,
        |                                          val:Cexp)
<prim-op>  ::= + | - | * | / | < | > | = | not |  eq? | string=?
                 | cons | car | cdr | list | pair? | list? | number?
                 | boolean? | symbol? | string?
<num-exp>  ::= a number token
<bool-exp> ::= #t | #f
<str-exp>  ::= "tokens*"
<var-ref>  ::= an identifier token
<var-decl> ::= an identifier token
<sexp>     ::= symbol | number | bool | string | ( <sexp>* )
```

באופן זה ניתן:

- ליצור מחלקה

```
(define pair
    (class (a b)
         ((first (lambda () a))
```

```
            (second (lambda () b))
            (sum (lambda () (+ a b)))
           )
     )
)

pair
→ Class
```

- ליצור אובייקט ממחלקה

```
(define p34 (pair 3 4))
p34
→ Object
```

- להפעיל מתודה של אובייקט

```
(p34 'first)
→ 3
(p34 'second)
→ 4
(p34 'sum)
→ 7
```

להלן עיקר המימוש של מבנה המחלקה (במודל ההצבה):

```
export type ClassExp = {tag: "ClassExp"; fields: VarDecl[]; methods: Binding[]; }
export type ClassValue = {
    tag: "Class";
    fields: string[];
    methodNames: string[];
    methodProcs : CExp[];
}
export type ObjectValue = {
    tag: "Object";
    methodNames: string[];
    methodProcs : Closure[];
}

const L3applicativeEval = (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) :
    isClassExp(exp) ? evalClass(exp) :
    isAppExp(exp) ? bind(L3applicativeEval(exp.rator, env), (rator: Value) =>
                        bind(mapResult(param =>
                            L3applicativeEval(param, env),
                              exp.rands),
                            (rands: Value[]) =>
                                L3applyProcedure(rator, rands, env))) :
    isLetExp(exp) ? makeFailure('"let" not supported (yet)') :

    makeFailure('Never');


const evalClass = (exp: ClassExp): Result<ClassValue> =>
    makeOk(makeClassValue(map(vd=>vd.var,exp.fields),
                          map(b=> b.var.var, exp.methods),
                          map(b=> b.val, exp.methods)));

const L3applyProcedure = (proc: Value, args: Value[], env: Env): Result<Value> =>
    isPrimOp(proc) ? applyPrimitive(proc, args) :
    isClosure(proc) ? applyClosure(proc, args, env) :
    isClassValue(proc) ? applyClass(proc,args,env) :
    isObjectValue(proc) ? applyObject(proc,args,env) :
    makeFailure(`Bad procedure ${format(proc)}`);

const applyClass = (cls: ClassValue, args: Value[], env: Env):
Result<ObjectValue> =>
    bind(mapResult(proc => L3applicativeEval(proc,env),cls.methodProcs),
        (methods: Value[]) =>
            allT(isClosure, methods) ?
                makeOk(makeObjectValue(cls.methodNames, map((method : Closure) =>
                   subFieldsIntoMethod(method, cls.fields, args), methods))) :
                makeFailure("Non-proc method"));

const applyObject = (object: ObjectValue, args: Value[], env: Env): Result<Value>
=> {
```

```
    if (args.length < 1)
       return makeFailure("Wrong number of parameters for method application");
     else {
            if (isSymbolSExp(args[0])) {
                const i = object.methodNames.indexOf(args[0].val);
                if (i >= 0) {
                    const proc : Closure = object.methodProcs[i];
                    if (proc.params.length != args.length - 1)
                        return makeFailure("Wrong number of parameters");
                    return applyClosure(proc,args.slice(1),env);
                } else return makeFailure(`Unrecognized method: ${args[0].val}`);
            } else
             return makeFailure("Wrong type of first argument for method");
        }
    }

const subFieldsIntoMethod = (method : Closure, fields : string[], args : Value[])
: Closure => {
    const proc : ProcExp = makeProcExp(method.params, method.body);
    const renamedProc : CExp = renameExps([proc])[0];
    const litArgs : CExp[] = map(valueToLitExp,args);
    const substituedProc : CExp = substitute([renamedProc], fields, litArgs)[0];
    return makeClosure(substituedProc.args, substituedProc.body) :
}
```

א. [10 נק'] ציינו מתי מחושבים המתודות של המחלקה והשדות של המחלקה, ומתי מתבצעת ההצבה של השדות ב-body של המתודות.

ב. [10 נק'] עדכנו את הקוד כך שהמתודות יחושבו בזמן יצירת המחלקה.

ג. [10 נק'] עדכנו את הקוד כך שהשדות יוצבו ב-body של המתודות רק בזמן הפעלתן.

ד. [5 נק'] איזו מהגישות עדיפה: זו שבמימוש הנוכחי, או זו שמוצעת בסעיפים ב-ג? נמקו בקצרה.

שאלה 2: טיפוסים [20 נקודות]

באלגוריתם היסק הטיפוסים שלמדנו בשיעור, השלב השלישי הוא יצירת משוואות טיפוס מתוך כללי היסק הנוגעים לביטויים ספציפיים ב-L5. כך למשל, מתוך כלל ההיסק עבור פרוצדורות בעלות פרמטר אחד לפחות נורה לאלגוריתם ליצור משוואה כך:

```
Given: (lambda (v1 ... vn) e1 ... em), construct the following equation:
```
T_{lambda} = [T_{v1} * ... * T_{vn} -> T_{em}].

כאשר T_{lambda} הוא משתנה הטיפוס עבור הביטוי `(lambda (v1 ... vn) e1 ... em)`.

בתרגיל 3 הכנסנו לשפה, בין היתר, את הביטויים union (איחוד טיפוסים) ו-diff (הפרש בין טיפוסים), את הביטוי any המבטא את קבוצת כל הטיפוסים, ואת פרדיקט הטיפוס שעבור טיפוס T יסומן is? T. לפרדיקט הטיפוס מעמד מיוחד על-פני בוליאן רגיל והוא משמש את מערכת בדיקת הטיפוסים.

א. [5 נק'] כתבו את כלל יצירת המשוואות, עבור ביטוי מסוג Type Predicate.

```
Given: (lambda (v) : is? T1 e1 ... em), construct the following equations:
```

ב. [5 נק'] כלל היסק שמשתנה בעקבות הכנסת הביטויים החדשים הוא הכלל ל-if. הסבירו באיזה אופן הכלל משתנה והשלימו את המשוואה הנוצרת. כאשר T_{if} הוא משתנה הטיפוס עבור הביטוי `(if test then alt)`.

```
Given: (if test then alt), construct the equation:
```
T_{if} =

בחזרה ל-L5 הרגילה. מכלל ההיסק עבור הפעלת פרוצדורה הורינו לאלגוריתם לייצר את המשוואה הבאה:

```
Given: (op a1 ... an), construct the following equation:
```
T_{op} = [T_{a1} * ... * T_{an} -> T_{app}]

כאשר T_{op} הוא משתנה הטיפוס עבור הביטוי `(op a1 ... an)`.

ג. [5 נק'] כתבו את כלל יצירת המשוואות עבור הצורה let.

```
Given: (let ((v1 c1) ... (vn cn)) e1 ... em), construct the following equations:
```

ד. [5 נק'] כזכור, את צורת let ניתן להמיר באופן דטרמיניסטי לביטוי של הפעלת פרוצדורה. האם משוואות שנוצרות מהפעלת פרוצדורה מעניקות יותר מידע על הטיפוסים מאשר משוואות על ביטוי let שקול? נמקו בקצרה.

שאלה 3: תכנות פונקציונאלי, CPS, רשימות עצלות [30 נקודות]

א. [6 נק׳] ציינו יתרון אחד וחיסרון אחד של שימוש ב-CPS:

ב. [6 נק'] ציינו שני מקרים בהם שימוש ברשימות עצלות הוא עדיף והסבירו מדוע.

ג. [6 נק'] ציינו שני מקרים בהם שימוש ברשימות עצלות הוא פחות עדיף והסבירו מדוע.

ד. [12 נק׳] ממשו את הפונקציה take-while$, אשר תיכתב בסגנון CPS, אשר מקבלת רשימה עצלה ופרדיקט, ומחזירה רשימה עצלה כל עוד האיברים מרשימת הקלט מקיימים את הפרדיקט.
במימוש זה אנו מניחים כי פרוצדורת התנאי pred היא פרימיטיבית.

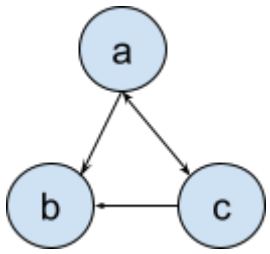
```
;; Type:
;;  [[T1 -> Boolean] * Lzl<T1> * [Lzl<T1> -> T2] -> T2]
;; Example:
;;  (take (take-while$ (λ (n) (< n 5)) (ints-from 0) (λ (x) x)) 10)
;;  => '(0 1 2 3 4))
```

שאלה 4: תכנות לוגי [15 נקודות]

ניתן לייצג גרף קשיר ומכוון בשתי דרכים:

- בעזרת הפרדיקט *edge/2* המציין צלע מכוונת בין שני קודקודים.
- על ידי רשימה של הצלעות

לדוגמא, ניתן לייצג את הגרף הבא בשתי דרכים:



- על ידי הפרדיקט edge/2

```
edge(a,b).
edge(a,c).
edge(c,b).
edge(c,a).
```

- על ידי רשימת הצלעות

```
[[a,b],[a,c],[c,b],[c,a]]
```

א. [5 נק'] ממשו את הפרוצדורה pred_to_list/1 הממירה את הייצוג הראשון בשני.

לדוגמא, עבור העובדות הבאות:

```
edge(a,b).
edge(a,c).
edge(c,b).
edge(c,a).

?-pred_to_list(L)
L = [[a,b],[a,c],[c,b],[c,a]]

?-pred_to_list([[a,b],[a,c],[c,b],[c,a]])
true

?-pred_to_list([[a,b],[f,g]])
false
```

במימוש ניתן להשתמש בפרוצדורה findall/3 של השפה Prolog, המחזירה רשימה של נתונים מתוך העובדות בתוכנית, על פי תבנית נתונה. לפרוצדורה שלושה ארגומנטים:

1. תבנית המידע אותו רוצים להחזיר
2. תבנית העובדות מהן המידע נשלף
3. רשימה מוחזרת של מידע המתאים לתבנית העובדות כך שכל איבר ברשימה מתאים לתבנית המידע.

לדוגמא:

```
parent(abraham, issac).
parent(abraham, yishmael).

?-findall(Son,parent(abraham,Son),L)
L = [issac, yishmael]
```

_____

_____

ב. [5 נק'] בשפה הלוגית Prolog קיים פרדיקט פרימיטיבי not, שערכו true אם הביטוי הלוגי שאליו הוא מתייחס ערכו false. לדוגמא:

```
male(abraham).

?- male(abraham).
true

?- male(sarah).
false

?- not(male(abraham)).
false

?- not(male(sarah)).
true
```

אחד הסטודנטים בקורס הציע להשתמש ב-not של פרולוג כדי לממש את pred_to_list באופן הבא:

```
pred_to_list(L):-pred_to_list([],L).

pred_to_list(Acc,Acc):-
    edge(N1,N2),
    member([N1,N2],Acc).
```

```
pred_to_list(Acc,L):-
    edge(N1,N2),
    not(member([N1,N2],Acc)),
    pred_to_list([[N1,N2]|Acc],L).
```

הסבירו בקצרה מדוע ערך השאילתא הבאה הוא אמת (אין צורך לצייר את עץ ההוכחה):

```
edge(a,b).
edge(a,c).
edge(c,b).
edge(c,a).

?-pred_to_list([[a,b]])
true
```

ג. [5 נק'] מה יהיה ערך השאילתא אם נהפוך את סדר החוקים בפרוצדורה pred_to_list/1?

```
pred_to_list(L):-pred_to_list([],L).

pred_to_list(Acc,L):-
    edge(N1,N2),
    not(member([N1,N2],Acc)),
    pred_to_list([[N1,N2]|Acc],L).

pred_to_list(Acc,Acc):-
    edge(N1,N2),
    member([N1,N2],Acc).


?-pred_to_list([[a,b]])
```