תאריך הבחינה: 5.7.2017 שעה: 09:00
שם המרצה: מני אדלר,מיכאל אלחדד, מירה בלבן, ירון גונן, דנה פישמן
מבחן בקורס: עקרונות שפות תכנות
מס' קורס: 202-1-2051
מיועד לתלמידי: מדעי המחשב והנדסת תוכנה
שנה: ב' סמסטר: ב'
מועד א'
משך הבחינה: 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) נא לוודא כי ב**שאלון** זה 13 עמודים
4) יש לענות על כל השאלות בגוף המבחן בלבד (בתוך השאלון). מומלץ לא לחרוג מהמקום המוקצה.
5) מומלץ להשתמש בטיוטא לפני כתיבת התשובה בתוך השאלון.
6) אם אינך יודע את התשובה, ניתן לכתוב "לא יודע" ולקבל 20% מהניקוד על הסעיף/השאלה.
7) הניקוד במבחן הוא כדלהלן:

שאלה 1: טיפוסים ופונקציות high-order	=	25 נק'
שאלה 2: רשימות עצלות	=	26 נק'
שאלה 3: דחיית חישוב ואופרטורים מיוחדים	=	20 נק'
שאלה 4: מודל הסביבות	=	12 נק'
שאלה 5: תכנות לוגי	=	22 נק'
---	---	---
סה"כ:	=	105 נק'

בהצלחה !!!

שאלה 1 [25 נק'] טיפוסים ופונקציות high-order

מבנה הנתונים Map ב-JavaScript ממש את הממשק הבא:

```
let m = new Map();
m.has(x):   determines whether m contains an entry <x,s(x)>
m.set(x, y): adds an entry <x,y> to the map m, or update <x,s(x)>
to
             <x,y> if such an entry exists.
m.get(x):   returns the value of x in m if it exists.
```

א. [3 נק'] השלם את הטיפוסים של הפונקציות בעזרת אנוטציות ב-TypeScript:

```
Map<T1,T2>:

has(key: T1): Boolean;

// assuming Map<T1,T2> is mutable
set(key: T1, value: T2): void;
// assuming Map<T1,T2> is immutable – this is also possible:
set(key: T1, value: T2): Map<T1,T2>;

get(key: T1): T2;
```

ב. [4 נק']

התבונן בהגדרה הבאה:

```
const memoize = (fn) => {
  let cache = new Map();
  return (x) => {
    if (!cache.has(x)) {
      cache.set(x, fn(x))
    }
    return cache.get(x)
  }
}

const fib = memoize((n:number) : number => {
    return  (n === 0) ? 0 :
            (n === 1) ? 1 :
            fib(n-1) + fib(n-2);
})
```

הסבר מה הפונקציה memoize מחשבת ומה היתרון שהפעלת memoize מעניקה להגדרת fib:

memoize is a higher-order function: it receives a function of one parameter and returns a closure of one parameter which remembers a map (x, fn(x)) for all arguments already computed. memoize improves runtime efficiency at the cost of memory consumption.
As a result, repeated invocations with the same parameter are computed only once.
In the case of fib, without memoize the runtime complexity is $T_n = T_{n-1} + T_{n-2}$ which is proportional to fib(n) ~ $O(1.6^n)$.
With memoize, the runtime complexity is O(n) since each call to fib(i) is performed only once.

Common errors:

- Answer both parts of the question (what does memoize do in general and how does its specific application to fib help)
- Indicate clearly that memoize constructs a closure which behaves in a specific way – memoize itself does not fill the cache – the closure that it computes does this.

ג. [5 נק'] השלם את הטיפוסים בהגדרת memoize:

```
const memoize : <T1,T2>(fn:(x:T1)=>T2) => ((x:T1)=>T2) =
<T1,T2>(fn:(x:T1)=>T2) : ((x:T1)=>T2)  => {
  let cache = new Map<T1,T2>();
  return (x:T1) : T2 => {
    if (!cache.has(x)) {
      cache.set(x, fn(x))
    }
    return cache.get(x)
  }
}
```

ד. [5 נק']

מגדירים פונקציות higher-order לעבודה מעל maps ב-TypeScript:

- **prop**: given a key, return an accessor function for a map that returns the value of this key.
- **propEq**: given a key and a value, return a predicate which tests whether a map has an entry <key,value>.

למשל:

```
const students = [
    {name: 'John', age: 23, dob: {year: 1994, month: 10, day: 10}},
    {name: 'Mary', age: 24, dob: {year: 1993, month: 07, day: 28}}
];

students.map(prop('age'));
==> [23, 24]

students.filter(propEq('name', 'John'));
==> [{name: 'John', age: 23, dob: {year: 1994, month: 10, day: 10}}]
```

השלם את הטיפוסים של הפונקציות ואת הקוד של propEq:

```
const prop : (key:string) => ((map:{}) => any) =
    (key: string) : ((map:{})=> any) => {
        return (map: {}) => map[key];
    }
```

Common errors:

- The function must return a function type

```
const propEq : <T>(key: string, value: T) => ((map:{}) => boolean) =
    <T>(key: string, value: T) => {
          return (map : {}) => map[key] === value;
    }
```

Common errors:
- The function must return a function type (map: {}) => Boolean
- The returned closure must test equality with "value".

ה. [8 נק]

מרחיבים את הפונקציה prop לפונקציה שיכולה לעבור על מסלול של מפתחות בתוך ערך מסוג map.

- **path**: Returns a function that when supplied an object returns the nested property of that object, if it exists.

כתוב את הפונקציה path והגדר עם טיפוסים מלאים.

```
students.map(path(['dob', 'year']))

==> [ 1994, 1993 ]
```

תזכורת: כאשר עובדים על מערך ב-JavaScript, ניתן להשתמש ב:

```
const arr = [1,2,3];
arr[0]       // ==> 1 (same as car)
arr.length   // ==> 3
arr.slice(1) // [2,3] (same as cdr) - returns a new array
```

```
const path : (keys: string[]) => ((map:{}) => any) =
    (keys: string[]) : ((map:{}) => any) => {
        return (map : {}) =>
            keys.reduce( (acc, curr) => acc[curr], map);
    }
```

Alternative solution without reduce:

```
const path : (keys: string[]) => ((map:{}) => any) =
    (keys: string[]) : ((map:{}) => any) => {
        if (keys.length === 0)
            return (map : {}) => map;
        else
            return (map:{}) => {
                let key0 = keys[0], rkeys = keys.slice(1);
                if (map[key0] === undefined)
                    return undefined;
                else
                    return path(rkeys)(map[key0]);
        }
    }
```

Common errors / Attention points:

- The return type must be a function from a map {} to any
- The return type is NOT a generic type T or an array T[] or a function to T[].
- The function must build a closure (map : {}) => …
- When keys is empty, the function behaves as identity (it is not an error)
- The function must traverse the map “down” (map[key[0]][key[1]]….)

שאלה 2 [26 נק'] רשימות עצלות

א. **[3 נק']** שימוש ברשימות עצלות

1. ציין, או הדגם, שני מקרים בהם רצוי להשתמש ברשימות עצלות. הסבר את היתרונות.
כאשר יש כמות לא מוגבלת של נתונים, וכאשר יש נתונים לא זמינים או שהגישה אליהם יקרה/סבוכה/לא בטוחה.

2. מהו החיסרון בשימוש ברשימות עצלות?
טיפוס נתונים חדש, דומה מאד לרשימות רגילות, דורש הכפלה של כל הפעולות. מקור נוסף לשגיאות טיפוס..
טעויות נפוצות: לא ניתן להגיע בצורה ישירה לאיברים ברשימה (גם ברשימה רגילה אי אפשר), שימוש מסורבל (יש ממשק, והעבודה דרכו היא כמו רשימות רגילות)

ב. **[7 נק']** למערכת ריאקטיבית יש מספר ערוצי קלט שבהם נקלטים אירועים (events) באופן המשכי. על מנת לתחזק את המערכת הוחלט להתממשק עם ערוץ קלט דרך רשימה עצלה, ולתחזק את המערכת כרשימה של רשימות עצלות לכל ערוצי הקלט. לצורך קליטת אירוע מפעילים פונקציה לקריאה מערוץ: `(read-channel i)` קוראת את האירוע הבא בערוץ `i`. לצורך הדוגמה, נניח שהאירועים המתקבלים מקריאת הערוצים הם מספרים.

```
(define make-event-feed
  (lambda (i)
    (cons-lzl (read-channel i) (lambda () (make-event-feed i)))))

> (make-event-feed 1)
; Channel 1: 1, 2…
'(1 . #<procedure:...>)

(define start-reactive-system
  (lambda (n)
    (map make-event-feed
         (enumerate-interval 1 n))))

> (start-reactive-system 3)
; Channel 1: 1, 2…
; Channel 2: 1, 2…
; Channel 3: 100, 200…
'((1 . #<procedure:...>)
  (1 . #<procedure:...>)
  (100 . #<procedure:...>))
```

המימוש שלעיל לא מאפשר להבחין, בהינתן ערך של אירוע, באיזה ערוץ האירוע הופיע. תקן את המידול של המערכת הריאקטיבית, כך שליד כל אירוע יצוין מספר הערוץ שבו הוא הופיע:

```
> (start-reactive-system 3)
; Channel 1: 1, 2
```

```
; Channel 2: 1, 2
; Channel 3: 100
'(((1 . 1) . #<procedure:...>)
  ((2 . 1) . #<procedure:...>)
  ((3 . 100) . #<procedure:...>))
```

לצורך התיקון, השתמש בפרוצדורה `lzl-map` למיפוי על רשימות עצלות:

```
(define start-reactive-system
  (letrec ((make-channel-event-feed
            (lambda (i) (lzl-map (lambda (e) (cons i e))
                        (make-event-feed i)))))
    (lambda (n)
      (map make-channel-event-feed
           (enumerate-interval 1 n))))
    )
```

טעויות נפוצות:

- העברה של רשימה רגילה כפרמטר ל־lzl-map , במקום רשימה עצלה.
- ערך חוזר של רשימה עצלה (במקום רשימה רגילה שכל איבר בה הוא רשימה עצלה)
- אי שימוש כלל ב־lzl-map (בניגוד להוראות מפורשות שכן להשתמש)

ג. **[6 נק']** רשימה עצלה יכולה לכלול מספר לא מוגבל (אינסופי) של נתונים.
רשימה היא **מוגדרת היטב** כאשר כל איבר של הרשימה ניתן לשליפה על ידי הפעלת מספר סופי של פעולות: לכל איבר **e** קיים **n** כך ש: **(nth lzl n) ➔ e**

מעוניינים לזהות, או, למצוא את ההופעה הראשונה של ערך כאירוע בערוץ כלשהו. לצורך זה, הוצע לאסוף את האירועים מכל הערוצים ולהפעיל פרוצדורת סינון `lzl-filter` על התוצאה. למשל:

```
> (lzl-filter (lambda (x) (= (cdr x) 2))
              (lzl-flatten (start-reactive-system 3)))
; Channel 1: 1, 3…
; Channel 2: 1, 2…
; Channel 3: 100, …
'((2 . 2) . #<procedure:...>)
```

לצורך זה, נכתבה פרוצדורה `bad-lzl-flatten` ל"שיטוח" רשימה של רשימות עצלות:

```
; Type: [List(Lzl) -> Lzl]
(define bad-lzl-flatten
  (lambda (lst)
    (if (empty? lst)
        empty-lzl
        (let ((lz1 (first lst))
```

```
            (rest-lzs (cdr lst)))
        (cond ((empty? rest-lzs) lz1)
              ((empty-lzl? lz1) (bad-lzl-flatten rest-lzs))
              (else
                (cons-lzl (head lz1)
                  (lambda () (bad-lzl-flatten
                               (cons (tail lz1) rest-lzs)))))))))
```

אבל הפעלתה על הנתונים שלעיל גרמה לולאה אינסופית:

```
> (lzl-filter (lambda (x) (= (cdr x) 2))
              (bad-lzl-flatten (start-reactive-system 3)))
; Channel 1: 1, 3, 4, 5, 6… (continue with increasing numbers)
; Channel 2: 1, 2, 3, 4, 5… (continue with increasing numbers)
; Channel 3: 100, 101, 102… (continue with increasing numbers)
ERROR: out-of-stack
```

הסבר למה הקריאה ל-**lzl-filter** לא מסתיימת:
הפרוצדורה bad-lzl-flatten יוצרת רשימה עצלה המורכבת קודם כל מכל האירועים בערוץ 1, שמספרם אינו חסום, והאירוע המבוקש, "2", לא הופיע בו.

ציין האם הרשימה העצלה הנוצרת על ידי פרוצדורת השיטוח היא מוגדרת היטב.
הרשימה אינה מוגדרת היטב מאחר ולא ניתן לשלוף את האירועים בערוצים אחרים.
טעות נפוצה מאוד: חזרה על הגדרת המושג "רשימה מוגדרת היטב" ללא הסבר מדוע הרשימה מתאימה להגדרה.

ד. [10 נק'] תקן את הפרוצדורה כך שתיצור רשימה עצלה מוגדרת היטב:

```
(define lzl-flatten
  (lambda (lst)
    (if (empty? lst)
        empty-lzl
        (let ((lz1 (first lst))
              (rest-lzs (cdr lst)))
          (cond ((empty? rest-lzs) lz1)
                ((empty-lzl? lz1) (lzl-flatten rest-lzs))
                (else
                  (cons-lzl (head lz1)
                    (lambda () (lzl-flatten
                                 (cons (second lst)
                                       (append (cdr rest-lzs)
                                               (list (tail lz1))))))))
                )))
    ))
```

שאלה 3 [20 נק'] דחיית חישוב ואופרטורים מיוחדים

א. **[3 נק']** דחיית חישוב היא טכניקה חישובית מקובלת. ציין שתי מטרות (הקשרים), שבהם משתמשים בדחיית חישוב.

בניית ערכים מורכבים אינסופיים, מניעת חישוב של ארגומנטים לפרוצדורות טרם שימושם, קריאה מ i/o באופן שאינו blocking, טיפול בלקוח בשרת כאשר יגיע תורו.

ב. **[3 נק']** ציין והדגם אופני דחיית חישוב בכל אחת מהשפות: **Scheme, JavaScript**

'עטיפת' חישוב ב-lambda (lambda abstraction), normal eval, CPS

Promise, callback ו generators

ג. **[2 נק']** מהו ההבדל בין אופרטור מיוחד **(special form)** לפרוצדורה פרימיטיבית?
הסבר והדגם בשפת **Scheme**.

לאופרטור מיוחד יש כלל חישוב מיוחד, כמו חישוב ביטויי if,define, השונה מדרך חישוב פרוצדורות (בפרט, לא כל תתי הביטויים בביטוי יחושבו בהכרח מראש).

פרוצדורה פרימיטיבית היא חלק מהפרימיטיבים של השפה, אך חישובה נעשה באותו אופן כמו שאר פרוצדורות משתמש (applicative/normal order)

ד. **[2 נק']** תארו את חוק ה חישוב של אופרטור המיוחד **OR** בשפת **Scheme**.

מדוע האופרטור איננו יכול להיות מוגדר כפרוצדורת משתמש ב-**applicative-eval**? הסבר והדגם.

בחוק החישוב של **or** מחושבים כל תתי-הביטויים עד אשר מתקבל ערך שאינו false. ערך זה הוא הערך של הביטוי כולו (ערך הביטוי אינו בהכרח #t). כלומר, לא נדרש לחשב את כל תתי הביטויים. ב applicative-eval יחושבו בכל מקרה כל הפרמטרים / תתי- הביטויים.

ה. **[2 נק']** אחד הסטודנטים בקורס טען כי אין צורך באופרטור מיוחד, עם חוק חישוב משלו, בשביל פעולה כמו **OR**, והציע את פרוצדורת המשתמש הבאה:

```
; Signature: or(val1, val2)
(define or
  (lambda (val1 val2)
    (if val1
        val1
        val2)))
```

האם הגדת ה-**or** (עם פרוצדורת המשתמש) תואמת את סמנטיקת חוק החישוב שתואר בסעיף ד'? נמק והדגם

לא. בפרוצדורה זו val1 ו val2 חושבו כבר, מתוקף היותם פרמטרים של פרוצדורה ב -applicative eval, בניגוד לחוק החישוב המיוחד של or שתואר לעיל:

- חישוב מיותר של val2 במידה ו val1 אינו false
- במידה ויש לחישוב val2 איזשהו side effect התוצאה תהיה אף שונה

ו. [4 נק']

הגדירו מחדש את פרוצדורת המשתמש **or**, כך שתתאים לסמנטיקת חוק החישוב בסעיף ג', תוך שימוש בטכניקה של דחיית חישוב. (אין לשנות את החתימה, אך ניתן לאלץ עם תנאי התחלה את אופי הפרמטרים):

```
;Signature: or(arg1,arg2)
;Type: [T1*T2 -> T1 | T2 | Boolean]
;Pre-condition: arg1 and arg2 evaluate to
;      <closure () T1 | T2 | Boolean>
(define or
  (lambda (arg1 arg2)
    (let ((arg1-val (arg1)))
      (if arg1-val
          arg1-val
          (arg2)))
          ))
```

ירדה נקודה לפתרון בו חושב val1 פעמיים.
ירדו שתי נקודות להבנה שצריך להפעיל closure עם מימוש שגוי
ירדו כל הנקודות לפתרונות ששינו את החתימה, סתם הגדירו lambda expression באופן מוזר, או התמקדו בהחזרת #t/#f

ז. [4 נק'] מהם החסרונות והיתרונות של הגדרת **OR** כאופרטור מיוחד, לעומת פרוצדורת משתמש:

1. אופרטור מיוחד:
   יתרון: אין תנאי-קדם מסוכן שיכול לגרום טעויות זמן ריצה. חיסרון –
   חיסרון: ריבוי אופרטורים מיוחדים

2. פרוצדורת משתמש:
   יתרון: אין כלל חישוב מיוחד. אין תלות במימוש מוקדם של האופרטור

   חיסרון: תנאי-קדם מסוכן שיכול לגרום טעויות זמן ריצה. אין אפשרות להבחנה משאר ביטויי תנאי

שאלה 4 [12 נק'] מודל הסביבות

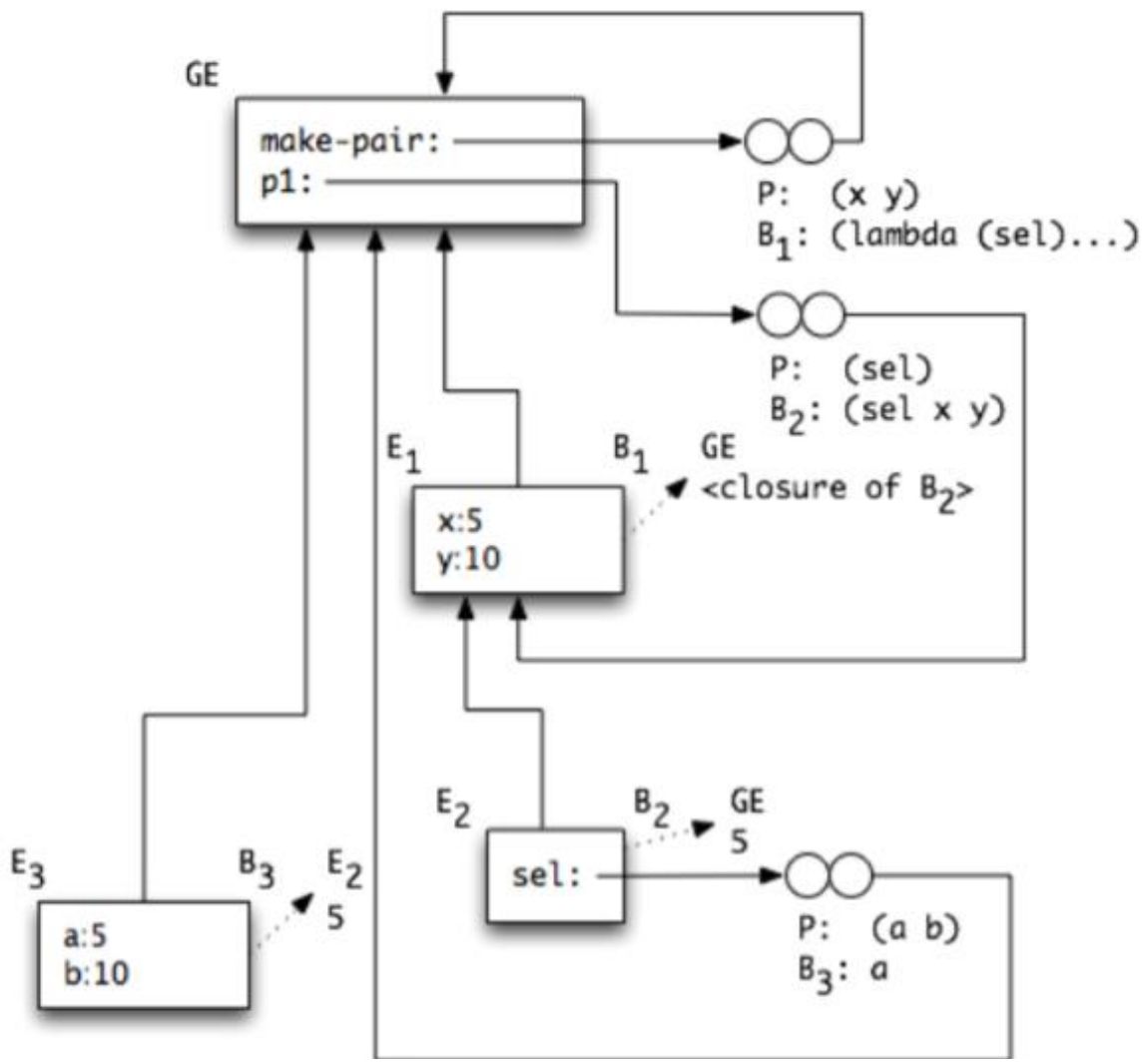
ניתנות ההגדרות למימוש מבוסס closures של מבנה נתונים pair:

```
(define make-pair
  (lambda (x y)
    (lambda (sel)
      (sel x y))))

(define p1 (make-pair 5 10))
```

השלם את דיגרמת הסביבות המתקבל בהרצה של הביטוי:

```
(p1 (lambda (a b) (+ a b)))
```



שאלה 5 [נק'] תיכנות לוגי

א. [6 נק'] לשני המקרים הבאים - חשב את הunifier של שתי הנוסחאות.
פרט את צעדי החישוב ואת התוצאה:

unify[t(X,f(a),X), t(g(U),U,W)]

1. s={} ,A = t(X,f(a),X) , B = t(g(U),U,W)
2. s={X=g(U)} , A°s= t(g(U),f(a), g(U)) B°s= t(g(U),U,W)
3. s={X=g(f(a)), U=f(a)} , A°s= t(g(f(a)),f(a), g(f(a))) B°s= t(g(f(a)), f(a),W)
4. s={X=g(f(a)), U=f(a), W= g(f(a)} , A°s= t(g(f(a)),f(a), g(f(a)))
   B°s= t(g(f(a)), f(a), g(f(a))

unify[t(X,f(X),X), t(g(U),U,W)]

1. s={} ,A = t(X,f(X),X) , B = t(g(U),U,W)
2. s={ X=g(U)} ,A = t(g(U),f(g(U)), g(U)) , B = t(g(U),U,W)
3. s={ X=g(U), U=f(g(U))} – OCCUR CHECK FAIL!

ב. [8 נק']

ניתנות הפרוצדורות הבאות המגדירות את קידוד מספרים טיבעיים בתיכנות לוגית:

```
% Signature: natural_number(N)/1
% Purpose: N is a natural number.
natural_number(0).                                  % N1
natural_number(s(X)) :- natural_number(X).          % N2

% Signature: Plus(X,Y,Z)/3
% Purpose: Z is the sum of X and Y.
plus(X, 0, X) :- natural_number(X).                 % P1
plus(X, s(Y), s(Z)) :- plus(X, Y, Z).               % P2

% Signature: times(X,Y,Z)/3
% Purpose: Z = X*Y
times(0, X, 0) :- natural_number(X).                % T1
times(s(X), Y, Z) :- times(X, Y, XY), plus(XY, Y, Z). % T2
```

הגדר את הפרוצדורות הבאות:

```
% Signature: exp(X,N,Z)/3
% Purpose: Z = X^N (power)
exp(X, 0, s(0)) :- natural_number(X).               % T1
exp(X, s(Y), Z) :- exp(X, Y, XY), times(X, XY, Z).  % T2

% Signature: fact(N,F)/2
% Purpose: F = N! (factorial)
fact(0, s(0)).                                      % T1
times(s(X),Y) :- fact(X,Z), times(s(X), Z, Y).      % T2
```

ג. [8 נק'] השלם את עץ ההוכחה החלקי שלהלן לחישוב כל התשובות. הקפד על שינוי שמות משתנים. על כל קשת בעץ, ציין מהי הצבת המשתנים עבורה, ומיהו הכלל שהופעל. ציין את התשובה בעלי הצלחה.