Question 1: CPS

Q1.b:

טענת עזר: $compose היא CPS שקולה ל compose.

הוכחת טענת העזר: נראה (compose$ f$ g$ pipe_cont) = (pipe_cont (f g)):

כתבנו את הפונקציה $compose כפי שנלמד בעמוד 96 ב lecture-notes במקרה בו יש הפעלה של פונקציה על פונקציה אחרת. בסיום נפעיל את pipe_cont על הרכבת הגרסה ה CPSית של הפונקציות.

טענה: $fact היא CPS שקולה ל fact. כלומר מתקיים:

(((pipe$ fs pipe_cont) x outer_cont) = (outer_cont (pipe_cont (pipe (fs x))

הוכחה: נראה באינדוקציה על אורך הרשימה fs:

בסיס: אורך הרשימה הוא 1. כלומר יש פונקצית אחת ברשימה fs. זהו מקרה הבסיס של $pipe ושל pipe:

a-e[((pipe$ fs pipe_cont) x outer_cont)] ⇒ a-e [(pipe_cont (lambda (x outer_cont) ((car fs) x outer_cont)))] ⇒ (car fs) is a CPS function ⇒ **a-e [(outer_cont (pipe_cont (car fs) x)))]]**

a-e[(outer_cont (pipe_cont (pipe (fs x)))] => **a-e [(outer_cont (pipe_cont (car fs) x)))]]**

צעד: הטענה נכונה עבור רשימת fs באורך n. כלומר נכונ למקרה ה alt של pipe:

a-e[((pipe$ fs pipe_cont) x outer_cont)] ⇒

a-e [(pipe$ (cdr fs) (lambda (res) (compose$ (car fs) res pipe_cont) x outer_cont)))))] ⇒

⇒ n קצרה מ (cdr fs) נפעיל את הנחת האינדוקציה כי

a-e[outer_cont(lambda (res) (compose$ (car fs) res pipe_cont) pipe((cdr fs) x)] ⇒

compose is CPS equivalent to compose$ ⇒

a-e[outer_cont(pipe_cont (compose (car fs) (pipe((cdr fs)) x)))] ⇒

⇒ compose נשערך את

a-e[outer_cont(pipe_cont (pipe(fs) x)))]

כנדרש

(לא חובה) דוגמת הרצה עבור (pipe$ (list add1$ square$) id) 3 id) :

נתבונן רק בפונקציה הנוצרת כתוצאה משיערוך של $pipe על רשימת הפונקציות ועל פונקציית ההמשך id:

אחרי הפעלה של $pipe:

(pipe$ (square$) (lambda (res) (compose$ add1$ res id)))

אחרי שיערוך של $compose:

(pipe$ (square$) (lambda (res) (id (lambda (x outer_cont) add1$ x (lambda (res1) (res res1 outer_cont)))))

אחרי עוד שיערוך של $pipe נגיע למקרה בסיס, כך שהפעם נשערך את הלמבדה המסומנת בירוק (היא ה pipe_cont בקוד):

(id (lambda (x outer_cont) add1$ x (lambda (res1) (res res1 outer_cont)))

במקרה זה res היא הארגומנט של pipe_cont במקרה הבסיס, שהוא:

(lambda (x outer_cont) (square$ x outer_cont))

סך הכל נקבל:

```
(id (lambda (x1 outer_cont1)
          (add1$ x1 (lambda (res1)
                    ((lambda (x2 outer_cont2) (square$ x2 outer_cont2))
                    res1 outer_cont1))))
```

בביטוי שמסומן בתכלת, נשערך את הלמבדה הכתומה כך ש x2=res1 ו outer_cont2=outer_cont1, אז נקבל:

```
(id (lambda (x1 outer_cont1)
                    (add1$ x1 (lambda (res1)
                              ((square$ res1 outer_cont1))
                              ))))
```

נבחין כי בדומה לעמוד 96 ב lecture-notes, הביטוי שקיבלנו הוא כמו הפעלה של add1, ואז square. תוצאה זו תהיה x מסוים, שעליו נפעיל את id.

Question 2: Lazy Lists

Q2.d:

reduce1-lzl: כאשר הרשימה סופית ונרצה לקבל את תוצאת הפעלת פונקציה על כל האיברים ברשימה. אם הרשימה אינסופית או גדולה מאוד, נרצה להשתמש ב reduce2-lzl.
reduce2-lzl: אם נרצה לקבל תוצאה בודדת של הפעלת פעולה על כל האיברים עד האיבר ה n. נשתמש כאשר לא נרצה לחשב את האיברים שאחרי האיבר ה n, אחרת נשתמש ב reduce3-lzl.
reduce3-lzl: אם נרצה לקבל רשימה עצלה בה כל איבר הוא תוצאה של פעולה על הכל האיברים עד אליו, כמו בסעיף f.

Q2.g:

יתרון של generate-pi-approximations: אם נרצה לדוגמא להדגים על גרף את התכנסות הסדרה לפאי, נוכל להדפיס כל נקודה ע"י איטרציה אחת (קריאה ל head) ב generate-pi-approximations, בעוד שב pi-sum נצטרך לבצע את כל החישוב מתחילת הסדרה בכל פעם.
יתרון נוסף הוא שב pi-sum הרקורסיה היא רקורסית ראש, כלומר כל קריאה רקורסיבית מחכה לחישוב מהקריאה הבאה, והקריאות ב generate-pi-approximations הן איטרטיביות.

חסרון של generate-pi-approximations: אם נרצה לחשב פעם אחת את הקירוב לפאי ונרצה לשמור רק את התוצאה, pi-sum תדרוש פחות זכרון מ generate-pi-approximations.

Question 3: Logic Programming

Q3.1: על סעיף זה נענה לפי "אלגוריתם למציאת MGU" בעמוד 112 ב lecture notes:

1. unify[x(y(y), T, y, z, k(K), y), x(y(T), T, y, z, k(K), L)] :

Case ג. Equations:
y(y) = y(T), T=T, y=y, z=z, k(K) = k(K), y=L

y(y) = y(T):
Case ג. Equations:
y=T, T=T, y=y, z=z, k(K)=k(K), y=L.

y=T:
Case ב. Equations:
T=y, y=y, z=z, k(K)=k(K), y=L.

T=T:
Equations:
y=T, z=z, k(K)=k(K), y=L.

z=z:
Equations:
y=T, k(K)=k(K), y=L.

k(K)=k(K):
Case ג. Equations:
y=T, K=K, y=L.

K=K:
Equations:
y=T, y=L.

MGU = {T=y, L=y}

2. unify[f(a, M, f, F, Z, f, x(M)), f(a, x(Z), f, x(M), x(F), f, x(M))]

Case ג. Equations:
a=a, M=x(Z), f=f, F=x(M), Z=x(F), f=f, x(M)=x(M).

a=a
Equations:
M=x(Z), f=f, F=x(M), Z=x(F), f=f, x(M)=x(M).

M=x(Z).
Case ב. Equations:
f=f, F=x(x(Z)), Z=x(F), f=f, x(x(Z))=x(x(Z)).

f=f.
Equations:
F=x(x(Z)), Z=X(F), f=f, x(x(Z))=x(x(Z)).

F=x(x(Z)).
Case ב. Equations:
Z=x(x(x(Z))), f=f, x(x(Z))=x(x(Z)).

Failure. Z is result of infinite x operations: Z=x(x(x(Z)))=(x(x(x(x(x(x(Z))))))=...

3. unify[t(A, B, C, n(A, B, C),x, y), t(a, b, c, m(A, B, C), X, Y)]

Case ג. Equations:

A=a, B=b, C=c, n(A,B,C)=m(A,B,C), x=X, y=Y

A=a.
Case ב. Equations:
B=b, C=c, n(a,B,C)=m(a,B,C), x=X, y=Y

B=b.
Case ב. Equations:
C=c, n(a,b,C)=m(a,b,C), x=X, y=Y

C=c.
Case ב. Equations:
n(a,b,c)=m(a,b,c), x=X, y=Y

Failure. n and m maybe not equal.

4. unify[z(a(A, x, Y), D, g), z(a(d, x, g), g, Y)]

Case ג. Equations:
a(A,x,Y)=a(d,x,g), D=g, g=Y.

a(A,x,Y)=a(d,x,g).
Case ג. Equations:
A=d, x=x, Y=g, D=g, g=Y

A=d.
Case ב. Equations:
x=x, Y=g, D=g, g=Y.

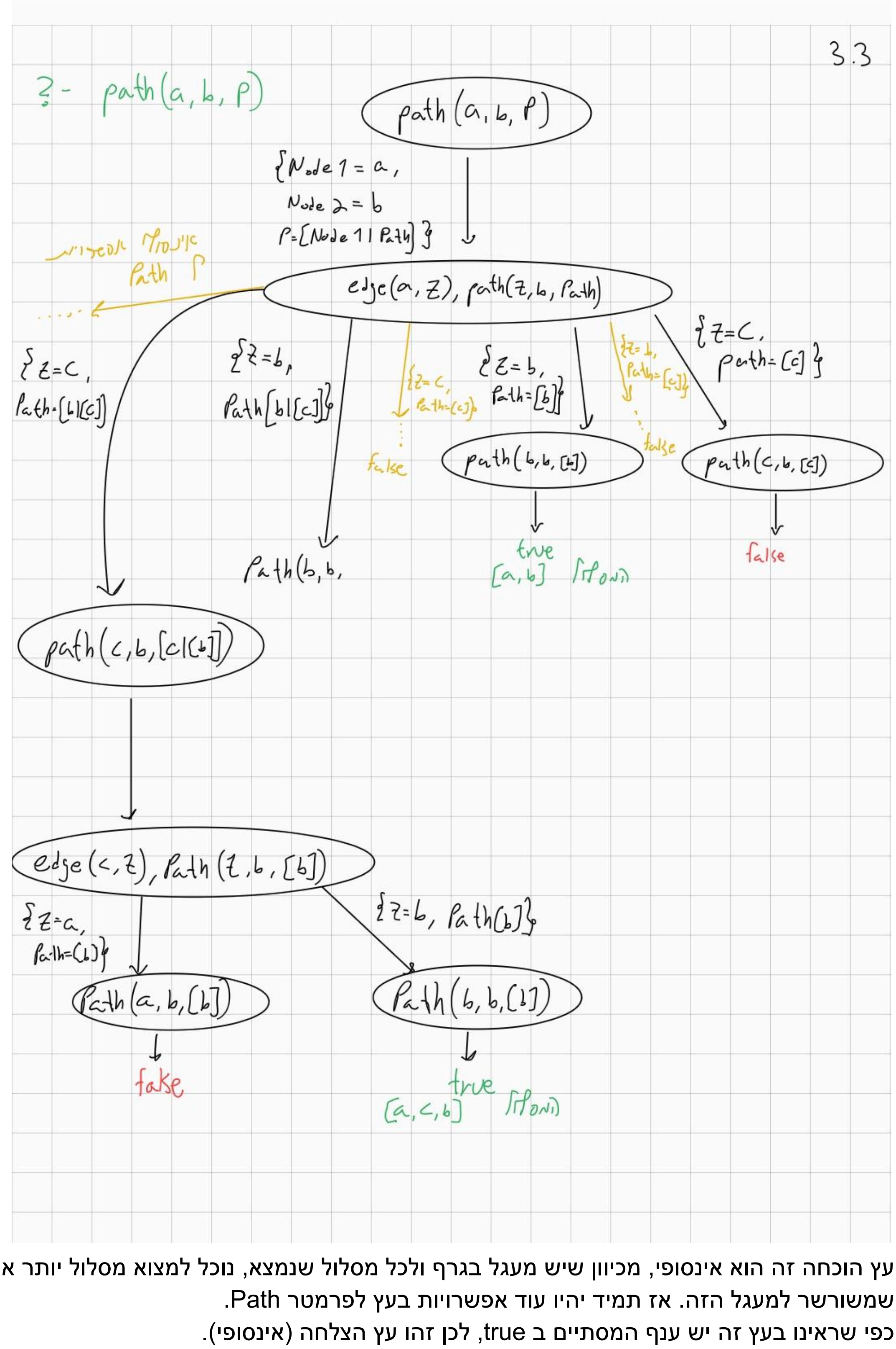
x=x.
Equations:
Y=g, D=g, g=Y.

Y=g.
Case ב. Equations:
D=g, g=g.

D=g.
Case ב. Equations:
g=g.

MGU = {A=d, Y=g, D=g}

Q3.3:



עץ הוכחה זה הוא אינסופי, מכיוון שיש מעגל בגרף ולכל מסלול שנמצא, נוכל למצוא מסלול יותר ארוך שמשורשר למעגל הזה. אז תמיד יהיו עוד אפשרויות בעץ לפרמטר Path.
כפי שראינו בעץ זה יש ענף המסתיים ב true, לכן זהו עץ הצלחה (אינסופי).

<u>הערה</u>: בקובץ של ה prolog השארנו למעלה את ההגדרות של צלעות ומספרים טבעיים תחת ההערה Defines