שפות תכנות - 236319

תרגיל בית 4 - חלק יבש

תאריך הגשה: 7.6.23

שם הסטודנט	ת.ז
מורן עאמר	322730789
עדן סרחאן	324849256

שאלה 1

בשאלה זו נדון בייצוגים אפשריים של מספרים טבעיים (כולל אפס) ב-SML וב-Lisp.

1. נציע את הייצוג הבינארי הבא למספרים הטבעיים:
- בייצוג יהיו שני סימבולים - "0" ו-"1".
- המספר אפס מיוצג על ידי רשימה המכילה את "0" בלבד, המספר אחד מיוצג על ידי רשימה המכילה את "1" בלבד.
- לכל 0<n - המספר n2 מיוצג על ידי הוספת "0" בראש הרשימה המייצגת את n, המספר n+12 מיוצג על ידי הוספת "1" בראש הרשימה המייצגת את n.

0 = [0]
1 = [1]
2 = [0,1]
3 = [1,1]
...

נרצה לממש את הפונקציה add שמקבלת ייצוגים של המספרים n ו-m ומחזירה את הייצוג של n+m. האם ניתן לממש זאת בשפת Lisp ובשפת SML? אם ניתן - הסבירו בקצרה איך. אם לא ניתן, נמקו למה.

פתרון:

ניתן לממש את הפונקציה add בשפת Lisp וגם בשפת SML

נסביר את המימוש בשתי השפות בו זמנית:
ניתן לממש את הפונקציה add באמצעות רקורסיה כאשר הפונקציה מקבלת שני ייצוגים של מספרים טבעיים m ו n ומבצעת חיבור בינארי שלהם.
נחלק למקרים:

1) במקרה של אחד מהם הוא אפס, אז הפונקציה מחזירה את הערך השני.
2) אחרת, היא מציבה את הסימבול הראשון בייצוג התוצאה להיות הסימבול המתאים לסכום של הסימבולים הראשונים בשני הייצוגים של n ו m, ומשתמשת ברקורסיה על הסימבולים הנותרים ובכך מבצעת חיבור סיבובי.

2. נציע את הייצוג האונארי הבא למספרים הטבעיים:
- המספר אפס מיוצג על ידי רשימה ריקה.
- לכל 0<n - המספר n מיוצג על ידי רשימה המכילה איבר בודד.
  האיבר הבודד הוא n-1 לפי אותו אופן ייצוג.

נרצה לממש את הפונקציה add שמקבלת ייצוגים של המספרים n ו m-
ומחזירה את הייצוג של n+m .
האם ניתן לממש זאת בשפת Lisp ובשפת SML ?
אם ניתן - הסבירו בקצרה איך. אם לא ניתן, נמקו למה.

0 = [ ]
1 = [0] = [[ ]]
2= [1] = [[[ ]]]
...

פתרון:

ניתן לממש את הפונקציה add בשפת Lisp ובשפת SML לא עבור הייצוג האונארי המתואר.

בשפת ,Lisp ניתן לממש את הפונקציה add באמצעות רקורסיה.
הפונקציה מקבלת שני ייצוגים של מספרים טבעיים n ,ו m-ומבצעת חיבור בינארי שלהם.
אם n הוא אפס, הפונקציה מחזירה את הערך השני.
אחרת, היא משתמשת ברקורסיה כדי לחשב את הסכום של n-1 ו m-1-
ונעביר את הרשימה שמכילה התוצאה הקודמת כתוצאה חדשה

בשפת ,SML אי אפשר , כי בSML נצטרך להיעזר ברקורסיה כדי לעבור על הרשימות n וm.
כלומר נצטרך לעבור על אותה פונקציה כמה פעמים.
הבעיה היא שעבור הפונקציה הזו נצטרך כל פעם לתת לפונקציה הזו טיפוסים שונים:
למשל עבור המספר 4 = [[[[[]]]]] :
באיטרציה הראשונה נתן לפונקציה הרקורסיבית 4, כלומר מטיפוס: a' list list list list list
באיטרציה השנייה נרצה לתת לפונקציה הרקורסיבית המספר 3 כקלט חדש, כלומר מטיפוס:
a' list list list list שזה קלט בעל טיפוס שונה
כלומר קוראים לאותה פונקציה עם טיפוס המשתנה בכל איטרציה ולכן אי אפשר לממש את הפונקציה.

ההבדל הוא שבLisp אין את הבעיה הנ"ל עם טיפוסים

3. נייצג ב-SML את סדרת המספרים הטבעיים כסדרה של פונקציות.
לפונקציה המייצגת את המספר n נקרא `nat_n`. הפונקציה `nat_n` מקבלת כפרמטר פונקציה f ומחזירה פונקציה שהיא הרכבה n פעמים של f:

$$nat\_n\ f = \overbrace{f \circ f \circ \cdots \circ f}^{n\ times}$$

למשל:

```
fun nat_0 _ = fn x => x;
fun nat_1 f = fn x => f x;
fun nat_2 f = fn x => f (f x);
fun nat_3 f = fn x => f (f (f x));
fun nat_4 f = fn x => f (f (f (f x)));
```

(א) הגדירו את הפונקציה nat2int שמקבלת פונקציה שמייצגת את המספר n ומחזירה את המספר n.

```
- nat2int nat_2;
val it = 2 : int
```

```
local

fun nat2int_aux nat_n f = nat_n f ;

fun plus x = x+1  ;

in

fun nat2int nat_n = nat2int_aux nat_n plus 0  ;

end;
```

(ב) הגדירו את הפונקציה nat2str שמקבלת פונקציה שמייצגת את המספר n ומחזירה מחרוזת של כוכביות באורך n.

```
- nat2str nat_3;
val it = "***" : string
```

```
local
   fun starsList ls 0 = ls
       | starsList ls n = starsList (("*")::ls) (n-1)

  fun starsString ls = foldr op^ "" ls;
in
  fun nat2str nat_n =
  let
     val n = nat2int nat_n;
     val ls = starsList [] n;
     val str = starsString ls;
  in
    str
  end;

end;
```

(ג) איזה ערך יוחזר מהקריאה הבאה? הסבירו בקצרה למה.

```
- nat2str (nat_2 nat_3);
```

ערך ההחזרה יהיה מחרוזת של 9 כוכביות.

הפונקציה nat_m nat_n הינה הרכבה של הפונקציה nat_n עם עצמה m פעמים.

כלומר נקבל : nat_n(nat_n(…))

נסמן את הפונקציה המתקבלת ע״י nat_k

ונקבל כי nat_k f היא הפעלת nat_k על עצמה m פעמים.

וזה יהיה הפעלת f על עצמה $n*n*n*\ldots*n=n^m$ פעמים.

במקרה הנתון n=3. m=2 נקבל כי זהו $nat\_3^2=nat\_9$

(ד) הפונקציה foo מקבלת שתי פונקציות שמייצגות את המספרים n ו-
m. איזה מספר מייצגת הפונקציה שהיא מחזירה? הסבירו בקצרה למה.

```
fun foo n m = fn f => fn x => m (n f) x;
```

הפונקציה foo מקבלת שתי פונקציות m ו n
והיא מחזירה פונקציה המקבלת f ו x
לפי המימוש הנתון של foo :

1) מפעילים n על f ומקבלים: nat_n f
2) מפעילים m על nat_n f ומקבלים : nat_m(nat_n f)

כלומר זה שווה להפעלת f על עצמה n פעמים ואז התוצאה מפעילים אותה על עצמה m פעמים.
סה"כ זהו הפעלת f על עצמה n*m פעמים כלומר nat_nm

(ה) הפונקציה bar מקבלת שתי פונקציות שמייצגות את המספרים n ו-
m. איזה מספר מייצגת הפונקציה שהיא מחזירה? הסבירו בקצרה למה.

```
fun bar n m = fn f => fn x => m n f x;
```

הפונקציה Bar מקבלת שתי פונקציות m ו n.
והיא מחזירה פונקציה המקבלת f. ו k.
לפי המימוש של Bar :

1) מפעילים nat_m על nat_n , לפי ההסבר בסעיף ג נקבל כי זהו nat_nm
2) מפעילים את התוצאה שקיבלנו ב- 1 על f

כלומר זה שווה להפעלת f על עצמה nm פעמים כלומר nat_nm. .

(ו) השלימו את הגדרת הפונקציה succ המקבלת כפרמטר פונקציה המייצגת את המספר n ומחזירה פונקציה המייצגת את המספר העוקב n+1.

```
fun succ n = fn f => fn x =>  f(n f x)
```

לדוגמה:

```
- nat2int (succ nat_2);
val it = 3 : int
```

(ז) השלימו את הגדרת הפונקציה add המקבלת שתי פונקציות המייצגות את המספרים n ו-m ומחזירה פונקציה המייצגת את n+m.

```
fun add n m = fn f => fn x => m f(n f x)
```

לדוגמה:

```
- nat2str (add nat_2 nat_3);
val it = "*****" : string
```

שאלה 2:

1. האם SML מאפשרת למתכנת להגדיר פונקציות מועמסות? אם כן, תנו דוגמה לפסקת קוד אשר מגדירה פונקציה מועמסת.

הערה :פונקציות פולימורפיות ב SML-אינן פונקציות מועמסות.

פתרון:

```
5 + 5; (* Val it = 10: int *)
1.5 + 2.3; (* Val it = 3.8: real *)
====> op+ is overloaded
```

בשפת SML לא מאפשרת לנו התכנתים להגדיר פונקציות מועמסות.
בשפת SML משתמשים בד"כ בפונקציות מועמסות וכפי שהוסבר בהערה זה לא זהה לפונקציות מועמסות.
במידה ונגדיר שתי פונקציות בעלות שם זהה, אז בעת ההרצה השפה תתיחס לפונקציה האחרונה שהגדרנו.
ניתן לתת לאותה פונקציה types (מספר סופי של טיפוסים) ולהבדיל ביניהם ע"י pattern matching אבל זה נחשב polymorphism לפי הגדרה.

2. האם ב SML-קיימת העמסה של מילה שמורה? אם כן, ציינו מהי ונמקו.

בשפת SML כן ישנה העמסה למילה שמורה למשל המילה of.
לדוגמא כאשר אנו רוצים להגדיר datatype המילה of יכולה לקבל כמה משמעויות שונות למשל bool, real,int ....
גם כן בתרגיל בית קודם שקיבלנו היה דוגמה להעצמה למילה השמורה bool :

```
datatype Truth = false | true | Maybe of bool;
val bool = true;
```

3. תנו דוגמה למזהה מעומס ב-SML. נמקו תשובתכם.

אפשר להעמיס לדוגמא את המזהה השמור bool. שהוא הטיפוס הבוליאני למספר מסוג int.

```
Val bool=0;
Bool +4;
```

ואז נקבל במשתנה התוצאה it את 4.

4. ידוע כי ב SML-קיים מנגנון אוטומטי לקביעת טיפוס פרמטר וערך החזרה של פונקציות. איך המנגנון מסתדר עם פונקציות מועמסות? האם שימוש בפונקציה מועמסת בתוך פונקציה אחרת יכול להפוך את האחרונה גם למועמסת?

ידוע לנו כי שפת SML לא מאפשרת העמסת פונקציות על ידי המשתמש.

לעומת זאת ישנן פונקציות מועמסות שהן מובנות בתוך השפה, למשל פונקציית print שיכולה לקבל כל מיני סוגים של משתנים להדפסה ולכן היא מועמסת.

המנגנון מסתדר עם פונקציות מועמסות ע"י הגדרת מה כל פונקציה מחזירה ואז הוא בוחר איזה פונקציה יבחר עם עדיפות לטיפוסים מסויימים על אחרים

דוגמה - op*:

```
fun f x y = x*y; (* Val f= fn: int->int->int *)
```

המנגנון נותן עדיפות לint על real ו string

אבל ניתן גם להשתמש באופרטור * על מחרוזות ומספרים ממשיים

השימוש בפונקציה מועמסת בתוך פונקציה אחרת אכן הופך את הפונקציה השניה גם למועמסת כי גם הפונקציה השניה עלולה להיות תלויה בפונקציה הראשונה (המועמסת) ולכן הפלט שלה משתנה בהתאם לטיפוסים.

שאלה 3

בהרצאה הוצגו לכם בנאים תיאורטיים שונים בעזרת שפת Mock. בשאלה זו נראה איך התיאוריה מתארת את המציאות.

עברו על המודול typing בשפה Python (כל המידע הנחוץ לשאלה מופיע בעמוד הנתון).

1. ציינו את בנאי הטיפוסים התיאורטיים (type constructors) שיש להם מקביל במודול typing לפי הסמנטיקה שמתוארת בקישור המצורף.

2. לכל בנאי שציינתם בסעיף הקודם הביאו דוגמה קונקרטית בפייתון לשימוש בו.

נפתור את שני הסעיפים ביחד:

- tuple types:

```
my_list = [2, 3, 5, 'a', 'b', 'c', True]
```

- Union types:

```
from typing import Union

# myVar accepts both integers and strings

myVar: Union[int, str]


myVar = 5

myVar = "Hello"
```

- Optional:

  From tyoing import Optional

  Def strlen (s: str) -> Optional[int]:

  If not s:

  Return None

  Return len(s)

- Callabl types:

```
class student:
   def greet(self):
      print("Hello there")

std = student()
print("Is student class callable? ",callable(student))
print("Is student.greet() callable? ",callable(std.greet))
print("Is student instance callable? ",callable(std))
```

- Set types:

  set1 = {"apple", "banana", "cherry"}
  set2 = {1, 5, 7, 9, 3}
  set3 = {True, False, False}

3. האם קיים מקביל בקישור המצורף לטיפוס התיאורטי bottom? אם כן מהו?

לא

** None type (bottom) is not the same as the optional types and the none type in Mock.

4. האם קיים מקביל בקישור לטיפוס התיאורטי top ? אם כן מהו?

כן, בקישור הנתון הטיפוס Any Type הינו הטיפוס המקביל , שמציג ערך בינארי שמאפשר להקצות לו ערך מכל סוג בשפה.

- Any type:

```
a: Any = None

s: str = ' '

a = 2     # (assign "int" to "Any")

s = a     # (assign "Any" to "str")
```