שפות תכנות - תרגיל בית 4:

מגישים:

אפק נחום 214392706

עידו טאוסי 214008997

שאלה 1:

1. כן, ניתן לממש את פונקציית *add* בשפת *SML* וב-*LISP*.

SML - נוכל להמיר את הרשימה שמייצגת את המספר הבינארי, למספר עשרוני בכך שנגדיר פונקציה רקורסיבית שמקבלת רשימה, המספר בבסיס עשרוני עד לקריאה זו ואינדקס איטרציה התחלתי, וקוראת לפונקציה (אם הרשימה ריקה נחזיר 0) עם הרשימה פחות הראש, המספר שחושב עד כה ועוד הערך של ראש הרשימה כפול (2 בחזקת אינדקס האיטרציה), ואינדקס האיטרציה ועוד 1. נשתמש בפונקציה הזו על מנת להמיר את 2 המספרים m,n שקיבלנו (בתור רשימות) מייצוג בינארי לעשרוני. לאחר שעשינו זאת, נסכום אותם באופן רגיל, ולאחר מכן נמיר את התוצאה מייצוג עשרוני לבינארי ע"י שימוש בפונקציה רקורסיבית שמקבלת מספר ומשרשרת את תוצאת mod 2 שלו לראש רשימה, והמשך הרשימה יהיה הפעלה של הפונקציה על חלוקה ב-2 של המספר. בדרך זו נייצר את התוצאה הרצויה מהפעולה.

LISP - נגדיר פונקציית *toDecimal* שמקבלת רשימה שמייצגת מספר בינארי, את המספר עד עכשיו, ואינדקס האיטרציה, ומחזירה את המספר הבינארי בבסיס עשרוני, נקרא לפונקציה מקוד המקור בצורה הבאה:(*toDecimal lst* 0 0)

וזהו פסאודו קוד של מימוש הפונקצייה:

```
(cond
((null lst) currentNum)
(T (toDecimal (cdr lst) (mul (power 2 i) (car lst)) + currentNum) (i + 1))))
```

בנוסף, כעת נצטרך לממש פונקציית *toBinary*, נעשה זאת באותה דרך כפי שעשינו ב-*SML*, בפסאודו קוד הבא:

```
(cond
((eq currentNum 0) Nil)
(T (cons(currentNum%2 toBinary (currentNum / 2)) ))
)
```

2. כן, ניתן לממש את פונקציית *add* בשפת *SML* וב-*LISP*.

SML - נבצע המרה מהייצוג הנ"ל לייצוג עשרוני, נבצע את החיבור ונמיר חזרה לייצוג הנתון. נוכל לממש המרה מהבסיס האונארי לבסיס העשרוני בכך שנגדיר פונקציה רקורסיבית המקבלת את הרשימה ומספר עד כה (בקריאה בקוד המקור יהיה 0). בפונקציה, כל עוד הרשימה אינה ריקה, נקרא לפונקציה מחדש, עם ראש הרשימה, והמספר עד כה ועוד 1, אם הרשימה ריקה, נחזיר את המספר עד כה. עבור פונקציית ההמרה מהבסיס העשרוני לבסיס האונארי נקבל את המספר שברצוננו להמיר, ואם המספר אינו 0, נקרא לפונקציה עם המספר פחות 1, ונשרשר את הפלט לרשימה ריקה, כלומר קיבלנו רשימה שהאיבר היחיד בה הוא הפלט מהפונקציה. אם המספר הוא 0, נחזיר רשימה ריקה.

LISP - נגדיר את אותן פונקציות שפועלות באופן זהה לפונקציות שהוגדרו ב-*SML*, בפסאודו קוד הבא:

toDecimal:

$$
\begin{gathered}
(cond \\
((null\ lst)\ currentNum) \\
(T\ (toDecimal\ (car\ lst)\ (currentNum + 1)\ )) \\
)
\end{gathered}
$$

toUnary:

$$
\begin{gathered}
(cond \\
((eq\ currentNum\ 0)\ ()\ ) \\
(T\ (cons\ (toUnary\ (currentNum - 1)\ )\ ()\ )) \\
)
\end{gathered}
$$

3.

א)

```
fun nat2int numFunc = (numFunc (fn x => x+1)) 0;
```

ב)

```
fun nat2str numFunc = (numFunc (fn x => x^"*")) "";
```

ג)

הערך שיוחזר מן הפונקציה הוא "*********", 9 כוכביות מפני שהפעלת (nat_2 nat_3) תחזיר לנו פונקציה המקבלת פונקציה f ומבצעת (nat_3 f) nat_3. במקרה של פונקציית nat2str, הפונקצייה המועברת היא הפונקציה המשרשרת כוכבית לסטרינג הנתון, ולכן nat_3 f הפנימי תרכיב אותה על עצמה 3 פעמים, ותחזיר פונקציה המשרשרת שלוש כוכביות לפרמטר הנתון, ואז nat_3 החיצונית עם פונקציה זו, תרכיב פונקציה זו 3 פעמים על עצמה, ולכן כאשר נפעיל את הפונקציה הסופית שקיבלנו בחזרה עם מחרוזת ריקה, נקבל מחרוזת המכילה 9 כוכביות, "*********".

ד)

נפתח את הביטוי, מהביטוי בסוגריים הפנימיים נקבל הרכבה של f כ-n פעמים על עצמה, וכעת נרכיב את התוצאה כ-m פעמים על עצמה ונקבל: $f^n \circ f^n \circ f^n \circ \dots \circ f^n(x) = \left(f^n(x)\right)^m = f^{m \cdot n}(x)$.
לכן המספר שתייצג הפונקציה שהיא מחזירה הוא $f^{m \cdot n}(x)$.

ה)

לפי אסוציאטיביות משמאל של שפת SML, נקבל כי הגדרת הפונקציה שקולה להגדרה הבאה:

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

כלומר נרכיב את nat_n על עצמה כ-m פעמים, ולכן בדומה לסעיף ג', התוצאה תהיה $f^{n^m}(x)$.

ו)

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

ז)

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

שאלה 2:

1. לא, בשפת SML מזהי הפונקציות צריכים להיות יחודיים, אחרת הקומפיילר יזרוק הודעה על שגיאת קומפילציה.

2. לא ניתן, המילים השמורות ב-SML לא ניתנות להעמסה.

3. = הוא מזהה מועמס ב-SML, למשל הוא משמש בתור אופרטור השמה ואופרטור השוואה.

4. המנגנון של השפה לא נותן להעמיס פונקציות, ולכן תיזרק שגיאת קומפילציה.

שאלה 3:

Mock	Python	Python example

Top	Any	`a: Any = None`
		`s: str = ''`
		`a = 2  #OK`
		`s = a  #OK`

Cartesian Product	Tuple	`def f(t: tuple[int, str]) -> None:`
		`  t = 1, 'foo'  # OK`
		`  t = 'foo', 1  # Error`

Mapping	Callable types (and lambdas)	`# next is a callable that gets int and returns int`
		`def twice(i: int, next: Callable[[int], int]) -> int:`
		`  return next(next(i))`

Disjoint Union	Union types	`def f(x: Union[int, str]) -> None:`
		`  if isinstance(x, int):`
		`  # x is int.`
		`  x + 1  # OK`
		`  else:`
		`  # x is str.`
		`  x + 'a'  # OK`

Unit Type	None type	`def f(x: int) -> None`
		`  print(x)`
		`  return None`

Branding	Type Aliases	`AliasType =`
		`Union[list[dict[tuple[int, str],`
		`set[int]]], tuple[str, list[str]]]`
		
		`def f() -> AliasType:`
		`  ...`

Records	Class types	`class A:`
		`  def f(self) -> int:`
		`  return 2`
		
		`def foo(a: A) -> None:`
		`  print(a.f())`

3. לא קיים טיפוס מקביל לטיפוס Bottom, מפני שאין אף טיפוס שמכיל 0 ערכים.

4. קיים טיפוס המקביל לטיפוס Top, וזה טיפוס Any.