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.

$LISP$ - ראשית, נאריך את n ו-m (לאורך $max\{len(n), len(m)\} + 1$) בייצוג הבינארי ע"י הוספת אפסים מימין בעזרת $cons$. נגדיר פונקציה ריקורסיבית calc שמקבלת 3 פרמטרים: n ו-m בייצוג הבינארי ו-carry, בכל איטרציה הפונקציה תבצע חיבור בין carry ו-2 הספרות שבראש הרשימות המייצגות של n ו-m. אם התוצאה של החיבור היא 2, נחזיר Cons של 0 עם רשימה שמכילה את הפלט של קריאה לcalc עם cdr של 2 הרשימות ו-carry=1. אם התוצאה היא 3 נחזיר Cons של 1 עם רשימה שמכילה את הפלט של קריאה לcalc עם cdr של 2 הרשימות ו-carry=1. אחרת, התוצאה היא 0 או 1 במקרה זה נחזיר של התוצאה עם רשימה שמכילה את הפלט של קריאה לcalc עם cdr של 2 הרשימות ו-carry=0. בצורה זו, אנו מבטיחים חיבור בצורה נכונה של ייצוג בינארי של 2 מספרים. תנאי העצירה של הפונקציה הוא כאשר n ו-m ריקות, במקרה זה נחזיר Nil ונסיים.

נשים לב שאין פעולת חיבור ב-lisp, על כן ניצור פעולת חיבור בין 3 מספרים שערכם 0 או 1 ע"י שימוש ב-eq. כלומר, נבדוק מה ערך של כל מספר ונחזיר בהתאם לכמות ה-1ים.

2. ניתן לממש את פונקציית add בשפת $LISP$ אך לא בשפת SML.

SML - בהינתן מספר בתצוגה אונארית, לא ניתן להמירו לייצוג עשרוני מפני שכל פונקציה רקורסיבית שנרצה לבנות על מנת להמירו למספר עשרוני, לא תוכל לפעול באיטרציה הבאה, מפני שנניח שקראנו לפונקציה עם הארגומנט [[]], כלומר רשימה של רשימה מטיפוס $'a$ כלשהו, כעת כשנרצה לקרוא לפונקציה שוב, עם ראש הרשימה, נרצה לקרוא לאותה פונקציה אך עם טיפוס ארגומנט שונה שהוא רשימה מאותו טיפוס $'a$ כלשהו []. לכן, הקוד לא יתקמפל מפני שבשפת SML לא ייתכנו טיפוסי ארגומנט שונים לאותה פונקציה. לכן לא ניתן "לפצח" את הארגומנטים m, n שקיבלנו ועל כן לא נוכל לחבר ביניהם וליצור את הייצוג האונארי של $m + n$.

$LISP$ - נגדיר פונקציה רקורסיבית המקבלת את שני המספרים בייצוגם האונארי, נסמנם m, n וכל עוד $n \neq nil$ נבצע קריאה לאותה פונקציה, כאשר במקום m יועבר $cons\ m\ ()$, כלומר עטפנו את m בעוד רשימה. במקום n יועבר $car\ n$, כלומר הסרנו מ-n עטיפה אחת של רשימה. לכן כאשר הפונקציה תקיים $n = nil$, סיימנו לחבר את המספרים ונוכל להחזיר את הארגומנט שבמקום m, מפני שהוא מכיל כעת את סכום המספרים בייצוג אונארי.

.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 הם יחודיים, אם נגדיר פונקציה שמקבלת ערכים מטיפוס מסוים, ואז ננסה לבצע העמסה בכך שנגדיר עוד פונקציה באותה שם שמקבלת טיפוסים שונים, תיזרק שגיאה מפני שברגע שניסנו להעמיס פונקציה, בפועל דרסנו את המימוש הקודם שלה, ולכן כשננסה לקרוא לה עם טיפוסי פרמטרים המתאימים למימוש שדרסנו, תיקרא שגיאה כי החתימה של הפונקציה החדשה שהגדרנו כבר לא תתאים.

2. בשפת SML קיימת העמסה של המילה השמורה in. מילה שמורה זו משומשת בבלוק $let \ldots \ in \ldots \ end$ ובנוסף גם בבלוק $local \ldots \ in \ldots \ end$. כאשר היא משומשת בבלוק $let \ldots \ in \ldots \ end$, ניתן להשתמש בכל מה שהוגדר בין $let \ldots in$ רק בבלוק $in \ldots end$. יתרה מזאת, כאשר היא משומשת בבלוק $local \ldots \ in \ldots \ end$, ניתן להשתמש בכל מה שהוגדר בין $local \ldots in$ רק בבלוק $in \ldots end$.

3. $=$ הוא מזהה מועמס ב-SML, למשל הוא משמש בתור אופרטור השמה ואופרטור השוואה נראה בדוגמה כי בשורה הראשונה הוא משומש להשמה של ערך, ובשורה השנייה הוא משומש להשוואה:

```
- val x=5;
val x = 5 : int
- 1=1;
val it = true : bool
```

4. המגנגנון של שפת SML לבדיקת טיפוסים יבדוק האם קיימת פונקציה המתאימה לטיפוס הרצוי עבור אחת מהפונקציות המועמסות. שימוש בפונקציה מועמסת בתוך פונקציה אחרת, יגרום לפונקציה שקוראת לפונקציה המועמסת לקבל את הטיפוס הדיפולטי של הפונקציה המועמסת, אם הוא לא יכול להסיק זאת מהפונקציה בעצמו, למשל אם היא מקבלת ושולחת במפורש טיפוס מסוים שהוא לא ברירת המחדל. אחרת, אם הוא לא יכול להסיק בעצמו, הוא יקבל את טיפוס ברירת המחדל, למשל בפונקציה $fun\ addNums\ x\ y = x + y;$, חתימתה תהיה:

```
- fun addNums x y=x+y;
val addNums = fn : int -> int -> int
```

למרות שאופרטור $+$ מקבל גם ערכים מטיפוס $real$, הטיפוס הדיפולטי שלו הוא int ולכן $addNums$ מוגדר עבור int בתור הטיפוס שהוא מקבל ומחזיר.

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	Named tuples	`Point = namedtuple('Point', ['x','y'])` `p = Point(x=1, y=2)`

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.

Memes


Game of life
ד"ר חיים
סטודנטים
בשפות
Hey, wait! Stop! I have עוד עבודה!
made with mematic