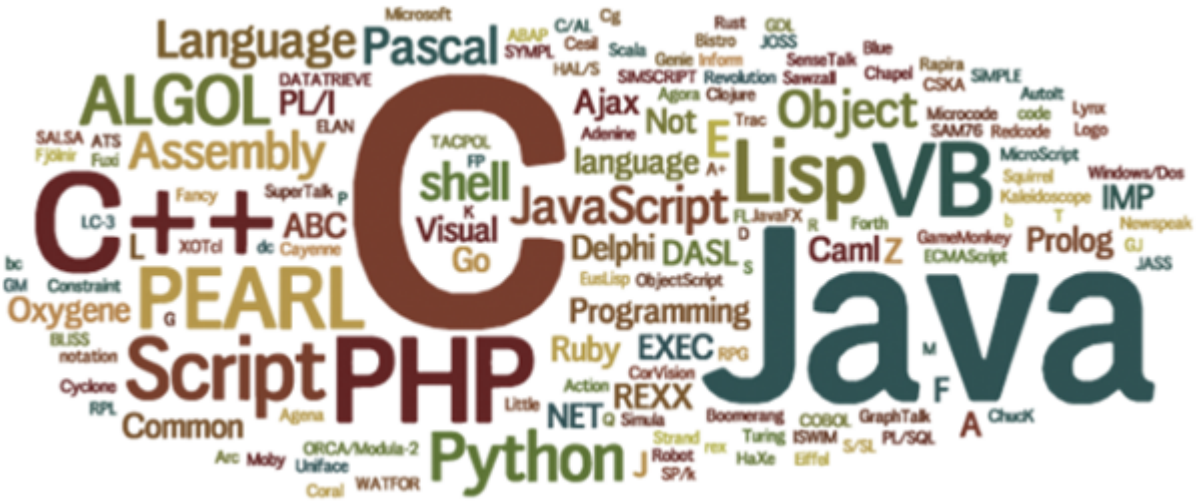
שפות תכנות, 236319

פרופ' ד. לורנץ

אביב 2023



תרגיל בית 3

תאריך פרסום: 4.5.2023

מועד אחרון להגשה: 21.5.2023 בשעה 23:55.

מתרגל אחראי: אנדריי בבין

מייל: andrey.babyn@campus.technion.ac.il

הערות:

- שאלות על התרגיל יענו בפורום הקורס בפיאצה **בלבד**.
- ניתן לפנות במייל במקרה של בעיות אדמיניסטרטיביות פרטניות, על כותרת המייל להתחיל ב-"*236319 - תרגיל בית 3*".
- הבהרות ותיקונים לתרגיל יפורסמו בגיליון זה ויסומנו בצהוב.
- לא תתאפשר הגשה לאחר מועד ההגשה כלל (אין הגשות באיחור).
- הסיבה היחידה שבגינה ניתן לקבל דחייה במועד הגשת תרגילי הבית היא מילואים. כדי לקבל דחיה עבור שירות מילואים, יש לפנות למתרגל האחראי על התרגיל במייל.
- ההגשה בזוגות **בלבד**. לפני ההגשה, ודאו כי ההגשה שלכם תואמת את הנחיות ההגשה בסוף התרגיל. **הגשות שלא יעמדו בהנחיות יפסלו על הסף**.

חלק יבש

לפניכם מספר שאלות, חלקן על דברים שראינו בכתה וחלקן לא. כתבו את הפתרונות אליהן בקובץ **dry.pdf**.

שאלה 1

1. נתון לכם הקוד הבא הכתוב בשפת SML:

```
fun trap_is_it senate =
  if List.hd senate = "Palpatine" then
  true orelse true else false andalso false;
datatype Truth = false | true | Maybe of bool;
val bool = true;
```

סווגו את כל מילה המופיעה בפסקת הקוד לפי המושגים הבאים:

- מילה שמורה/מילת מפתח
- מזהה שמור
- מזהה מוגדר מראש
- מזהה המוגדר על ידי המשתמש
- מזהה של ספריה
- ליטרל
- אחר

הערות:

- כל מזהה חוקי ב-SML נחשב כמילה בשאלה זו.

2. מבין המזהים אשר **הוגדרו על ידי המשתמש**, ציינו מהן הישויות שהם מסמנים. כגון, פונקציה, ערך מספרי וכדומה.

שאלה 2

שאלה זו מתייחסת ל[מפרש לשפת Lisp הממומש בשפת Lisp](#) שראינו בהרצאה.

1. סמנו ב-X את התשובה הנכונה בטבלה.

מה מוחזר עבור כל אחת מהקריאות הבאות ל-EVAL?	NIL	T	A	שגיאה
`(EVAL '(CAR NIL) '())`				
`(EVAL '(COND) '())`				
`(EVAL '(COND (NIL (QUOTE A)) (T (CAR NIL))) '())`				
`(EVAL '(COND (T (QUOTE A)) (NIL (CAR NIL))) '())`				
`(EVAL '(COND (NIL (QUOTE A)) (NIL (CAR NIL))) '())`				
`(EVAL '(COND (T (QUOTE A)) (T (CAR NIL))) '())`				

2. הוסיפו למימוש של **EVAL** תמיכה בביטוי **LET** מהצורה הבאה:

```
(LET nameExp valueExp bodyExp)
```

ביטוי זה יוצר binding בין **nameExp** ל-**valueExp** שאפקטיבי רק ב-**bodyExp**.
דוגמה לשימוש:

```
>>> (EVAL '(LET (QUOTE X) (QUOTE A) (CONS X NIL)) '())
(A)
>>> (EVAL '(LET (QUOTE X) (QUOTE A) (LET (QUOTE Y) (QUOTE B) (LIST X Y))) '())
(A B)
```

כתבו את הקוד שיש להוסיף למפרש וציינו בין אילו שורות יש למקם אותו.

3. הוסיפו למימוש של EVAL תמיכה בביטוי ORELSE. הביטוי

$$(\mathbf{ORELSE}\ boolExp_1\ boolExp_2)$$

בדומה ל-orelse ב-SML, שיערוך הביטוי מחזיר **T** במקרה ואחד הביטויים משתערך לערך אמת. שימו לב, כמו ב-SML, שיערוך הביטוי הוא בצורת short-circuit. למשל, שיערוך הביטוי הבא יחזיר **T** ולא שגיאה. דוגמה לשימוש:

```
>>> (EVAL '(ORELSE (QUOTE A) (CAR NIL)) '())
T
```

כתבו את הקוד שיש להוסיף למפרש וציינו בין אילו שורות יש למקם אותו.

שאלה 3

נייצג **רשימה** ב-Lisp על ידי datatype ב-SML באופן הבא:

```
datatype 'a Sexp = Atom of 'a | List of 'a Sexp list
```

לדוגמה, הרשימה `(cons hello (World))` ב-Lisp תיוצג ב-SML על-ידי:

```
List[Atom "cons", Atom "hello", List[Atom "World"]]
```

הגדירו את 4 הפונקציות הבאות (הניחו כי הקלט חוקי):

```
val cons = fn : 'a Sexp * 'a Sexp -> 'a Sexp
val car = fn : 'a Sexp -> 'a Sexp
val cdr = fn : 'a Sexp -> 'a Sexp
val toString = fn : string Sexp -> string
```

דוגמת הרצה:

```
- val x = List[Atom "cons",Atom "hello", List[Atom "World"]];
- toString x;
val it = "(cons hello (World))" : string

- toString (cons (Atom "hi", x));
val it = "(hi cons hello (World))" : string

- toString (car x);
val it = "cons" : string

- toString (cdr x);
val it = "(hello (World))" : string

- toString (cdr (cdr x));
val it = "((World))" : string

- toString (cdr (cdr (cdr x)));
val it = "()" : string
```

חלק רטוב

- קבצים המסופקים לחלק זה תוכלו למצוא בגיטהאב של הקורס - Homework3.
- פתרון לכל אחת מהשאלות יש לכתוב בקובץ נפרד ששמו hw3_q<i>.sml עבור שאלה i. כמו כן, יש להוריד קובץ הגדרות לכל שאלה ולייבא אותו על ידי הוספת השורה הבאה בתחילת הפתרון:

```
use "hw3_q<i>_def.sml";
```

- לכל שאלה מסופקת לכם דוגמת קלט ופלט רצוי. לצורך העניין את הפתרון שלכם לשאלה 1 תוכלו לבדוק בעזרת הפקודות הבאות:

```
sml hw3_q1.sml < q1.in > q1.out
diff <(sed '1,/===TEST START===/d' q1.out) q1.expected
```

- יש לבדוק ולהריץ את פתרונותיכם לתרגיל זה בדוקר **twyair/safot-hw:3**
- מדריך התקנה ושימוש בדוקר

שאלה 1

ד"ר חיים מהפקולטה לביולוגיה נלהב מהמימוש הנהדר של הפונקציה **is_alive** שכתבתם בתרגיל הבית הקודם. הוא מזמין אתכם להמשיך לעבוד איתו על הסימולטור.

אך, בניגוד לסטודנטים של הפקולטה שלנו אשר מסתפקים בתצוגה של המידע בטרמינל או דיבאגר, הסטודנטים של ד"ר חיים מעדיפים תצוגה ויזואלית. לפיכך אתם מתבקשים לממש "ממשק" גרפי ב-SML.

המשימה הזו הרגישה לכם קלה מדי ורציתם להציג את היוקרה של הפקולטה. על כן יזמתם לממש את כל הפונקציות של הממשק כ-**one-liners**! כלומר עליכם לממש את כל הפונקציות בשאלה הזו **ללא** שימוש בביטויי תנאי (if), ב-pattern matching (כולל case of), קריאות רקורסיביות, refs, ופונקציות עזר (אין להשתמש גם ב-let ו-local).

נגדיר שני מבנים הבאים:

- **frame** או מסגרת - רשימה של מחרוזות באורך שווה המורכבות רק מרווחים וכוכביות. כל מחרוזת ברשימה מייצגת שורה בתמונה כאשר כוכבית מייצגת פיקסל דלוק ורווח פיקסל כבוי. דוגמה למסגרת המייצגת סמיילי:

```
type frame = string list;
val smile_frame = [
"   *  *   ",
"   *  *   ",
" *      * ",
"  ******  "
];
```

- **state** או מצב - רשימה של רשימות באורך שווה של ערכים בוליאנים. מצב הינו הייצוג של המסגרת בצורה נוחה לעבודה של המתכנת. דוגמה למצב אשר מתאר את הסמיילי:

```
type state = bool list list;
val smile_state = [[false, false, false, true, false,
false, true, false, false, false], [false, false, false,
true, false, false, true, false, false, false], [false,
true, false, false, false, false, false, false, true,
false], [false, false, true, true, true, true, true, true,
false, false]];
```

כפי שניתן לשים לב, מצב הוא ייצוג פחות קריא.

הערה: חתימות של הפונקציות לא יבדקו ולכן אין צורך להכריח הדפסה של frame במקום string list או state במקום bool list list בחתימום.

1. ממשו את הפונקציה **mapState** אשר מקבלת פונקציה ומצב ומחזירה רשימה של רשימות של הערכים המתקבלים מהפעלת הפונקציה על תאי המצב.

```
mapState: (bool -> 'a) -> state -> 'a list list
```

דוגמת הרצה:

```
- mapState (fn true => 1 | false => 0) [[false, true,
false], [true, false, true]];
val it = [[0, 1, 0], [1, 0, 1]] : int list list
```

2. ממשו את הפונקציה **toString** אשר מקבלת רשימה של תווים ומחזירה מחרוזת המורכבת מהתווים.

```
toString: char list -> string
```

דוגמת הרצה:

```
- toString (explode "Hello world");
val it = "Hello world": string
```

הערה: אין להשתמש בפונקציה implode או בכל פונקציה אחרת שלא למדנו.

3. ממשו את הפונקציה **frameToState** אשר מקבלת מסגרת ומחזירה מצב המתאר את המסגרת.

```
frameToState: frame -> state
```

דוגמת הרצה:

```
- frameToState [" * ", "* *"];
val it = [[false, true, false], [true, false, true]] :
state
```

הערה: ניתן להשתמש בפונקציה explode אשר מקבלת מחרוזת ומחזירה רשימה של תווים מהם מורכבת המחרוזת.

4. ממשו את הפונקציה **stateToFrame** אשר מקבלת מצב ומחזירה מסגרת המתאימה למצב. ניתן להיעזר בפונקציה **toChar** המסופקת לכם.

```
stateToFrame: state -> frame
```

דוגמת הרצה:

```
- stateToFrame [[false, true, false], [true, false, true]];
val it = [" * ", "* *"] : frame
```

5. ממשו את הפונקציה **printFrame** אשר מקבלת מסגרת ומדפיסה אותה על המסך.
יש לבצע ירידת שורה אחרי כל שורה במסגרת.

```
printFrame: frame -> unit
```

דוגמת הרצה:

```
- printFrame [" * ", "* *"];
 *
* *
val it = () : unit
```

בסעיף זה עליכם להיעזר בפונקציה **print** המובנת בשפה אשר מדפיסה מחרוזת על המסך.

```
print: string -> unit
```

הערות:

- את כל חמש הפונקציות יש לממש כ-one-liners.
- בכל פונקציה ניתן להשתמש בפונקציות מהסעיפים הקודמים.
- יש לייבא את קובץ ההגדרות על ידי השורה:

```
use "hw3_q1_def.sml";
```

שאלה 2

בשאלה זו נרצה לממש מודולים להפעלה של פונקציות גרעין (קרנל) חד ודו-מימדיות.

מודולים אלו ישמשו את ד"ר חיים.

הגדרות:

פונקציית גרעין (חד מימדית) - פונקציה המקבלת שלושה איברים מאותו טיפוס ומחזירה איבר בודד. לדוגמה:

```
- fun sum leftNeighbor value rightNeighbor =
            leftNeighbor + value + rightNeighbor;
val sum = fn : int -> int -> int -> int
```

פונקציית הפעלה - פונקציה המקבלת רשימה (בלבד) ומחזירה רשימה. איברי רשימת הפלט מתקבלים על ידי הפעלת פונקציית גרעין ידועה מראש עבור כל איבר ברשימת הקלט, יחד עם שני שכניו.

- במידה ואין שני שכנים (איברי קצה), יועבר ארגומנט דיפולטיבי, גם הוא ידוע מראש.

דוגמה להרצת פונקציית הפעלה אשר מוגדרת בעזרת פונקציית הגרעין sum ומשתמשת בערך ברירת מחדל 0:

```
- runKernelSum [1, 2, 3, 4];
val it = [3, 6, 9, 7] : int list
```

פונקציית גרעין דו-מימדית - פונקציית גרעין אשר שלושת האיברים אותה מקבלת הם מטיפוס tuple המכיל שלושה איברים. כל האיברים מכל ה-tuples (סך הכל תשעה) הם מאותו טיפוס. לדוגמה:

```
- fun weird_sum (x1, x2, x3) (y1, y2, y3) (z1, z2, z3) =
    2 * x2 + y2 + 3 * z2;
val weird_sum = fn :
    'a * int * 'b -> 'c * int * 'd -> 'e * int * 'f -> int
```

הפעלת פונקציית גרעין דו-מימדית מוגדרת באותו אופן כמו חד מימדית, פרט לכך שפועלת על רשימה של רשימות (ראו דוגמה בסעיף 3).

1. בקובץ ההגדרות מסופקת לכם חתימה של גרעין חד-מימדי בשם **KERNEL1D_SIG**. עליכם לממש functor בשם **Kernel1D**. ה-functor יקבל מודול אשר מממש את החתימה ומחזיר מודול המכיל פונקציית הפעלה בודדת בשם **runKernel**.

```
runKernel: source list -> target list
```

דוגמת הרצה:

```
- structure SumKernel = Kernel1D(struct
    type source = int
    type target = int
    val kernel = sum
    fun default _ = 0
end);
- SumKernel.runKernel [1, 2, 3, 4];
val it = [3, 6, 9, 7] : int list
```

תוכלו להסיק את המשמעויות של ההצהרות שונות בעצמכם מהדוגמה ומהקבצים המסופקים.

.2

a. ממשו את הפונקציה **zip** המקבלת שלוש רשימות בגודל זהה ומחזירה רשימה של tuples המורכבים מהאיברים המתאימים מכל רשימה.

```
zip: 'a list -> 'b list -> 'c list -> ('a * 'b * 'c) list
```

דוגמת הרצה:

```
- zip [1, 2, 3] [4, 5, 6] [7, 8, 9];
val it = [(1, 4, 7), (2, 5, 8), (3, 6, 9)] : int * int *
int) list
```

יש לזרוק חריגה בשם **SizeNotEqual** במקרה והרשימות אינן באותו גודל.

b. ממשו את הפונקציה **fill** אשר מקבלת ערך כלשהו ומספר **i**. הפונקציה תחזיר רשימה המכילה את הערך הנתון **i** פעמים.

```
fill: 'a -> int -> 'a list
```

דוגמת הרצה:

```
- fill false 3;
val it = [false, false, false];
```

יש לזרוק חריגה בשם **NegativeLength** במידה ומתקבל מספר שלילי.

3. בקובץ ההגדרות מסופקת לכם חתימה בשם **KERNEL2D_SIG**. עליכם לממש functor בשם **Kernel2D** המקבל מודול העומד בתנאי חתימה זו. המודול אותו יצור ה-functor יכיל פונקציה בודדת בשם **runKernel** המפעילה פונקציית גרעין **דו-מימדית** הנתונה על ידי מודול הקלט.

```
runKernel: source list list -> target list list
```

דוגמת הרצה:

```
- structure SumKernel = Kernel2D(struct
    type source = int
    type target = int
    fun kernel (x1, x2, x3) (y1, y2, y3) (z1, z2, z3) = 2 *
x2 + y2 + 3 * z2
    fun default _ = 0
end);
- SumKernel.runKernel [   [1, 2, 3],
                          [4, 5, 6],
                          [7, 8, 9]    ];
val it = [[7, 13, 7], [19, 31, 16], [31, 49, 25]] : int
list list
```

שימו לב כי פונקצית גרעין קודם מקבלת את **העמודה** הראשונה ולא את השורה.

רמז: ניתן לממש את ה-functor הדו-ממדי על ידי הפעלה פעמיים של ה-functor החד-ממדי. פעם אחת עם גרעין zip ופעם אחת עם גרעין הרצוי.

הנחיות הגשה

- את הפתרון לתרגיל הבית יש להגיש בקובץ zip המכיל את הקבצים הבאים בלבד:

hw3_q1.sml, hw3_q2.sml, dry.pdf.

- על החלק היבש להיות מוקלד. אין להגיש סריקה או צילום של התשובות לחלק זה.
- שם קובץ ההגשה יהיה EX3_ID1_ID2.zip כאשר ID1, ID2 הם מספרי ת.ז. של המגישים.
- מסופק לכם סקריפט לבדיקת עצמית של קובץ הגשה בשם **selfcheck.sh**. אנא בדקו את תקינות ההגשה שלכם בעזרת הסקריפט בסביבת ה-docker.
- בודקי התרגילים **מאוד** אוהבים Memes. שתפו את תחושותיכם במהלך פתירת התרגיל באמצעות Memes מתאימים ב-pdf של החלק היבש. Memes מצחיקים בצורה יוצאת דופן יזכו את היוצרים בבונוס.

בהצלחה!