18.03.2021

מבחן סוף סמסטר – מועד ב׳

מרצה אחראי: ד״ר שחר יצחקי

מתרגלים: מתן פלד, איתן סינגר, עומר בלחסין

הוראות:

א. בטופס המבחן 13 עמודים מהם 6 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן שעתיים וחצי (150 דקות).

ג. אסור כל חומר עזר חיצוני.

ד. במבחן 4 שאלות. כל השאלות הינן חובה. משקל כל שאלה מופיע בראשיתה. (חלוקת המשקל בין הסעיפים בכל שאלה אינה בהכרח אחידה.)

ה. ניתן לציין לגבי סעיף או שאלה ״לא יודע/ת״. תשובה זו תזכה ב- 20% מהניקוד של הסעיף או השאלה. תשובות שגויות לא יזכו בניקוד.

ו. חובה לנמק כל תשובה. לא יינתן ניקוד על תשובות ללא נימוק.

ז. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.

ח. אין צורך להגיש את הטופס בתום הבחינה.

ט. את התשובות לשאלות יש לרשום במחברת הבחינה בלבד.

בהצלחה!

שאלה 1: שלבי הקומפילציה ואופטימיזציות (25 נק׳)

חלק א׳: סיווג מאורעות (10 נק׳)

נתונה התוכנית הבאה בשפת FanC:

```
1   int foo(int n) {
2     int c = 1;
3     while (c != 0) {
4       c = c - 1;
5       if (n > 100) {
6         n = n - 10;
7       } else {
8         n = n + 11;
9        c = c + 2;
10      }
11    }
12    return n;
13  }
14  int bar(int x) {
15    x = x * 2 + 2;
16    if (x > 50) {
17      return x - 50;
18    } else {
19      return x + 50;
20    }
21  }
22  void main() {
23    int acc = 0;
24    int i = 0;
25    while (i < 500) {
26      acc = acc + foo(bar(16 + 32));
27    }
28  }
```

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) לקוד של התוכנית, או מאורע שמתרחש בזמן השימוש בקוד.
שלבי הקומפילציה הם: ניתוח לקסיקלי, ניתוח תחבירי, ניתוח סמנטי, ייצור קוד, זמן ריצה.
הניחו כי התוכנית לפני השינויים מתקמפלת ורצה ללא שגיאות. עבור כל שינוי כתבו האם הוא גורם לשגיאה, ואם כן, ציינו את השלב המוקדם ביותר שבה נגלה אותה ונמקו בקצרה:

א. (2 נק׳) מחליפים את שורה 24 ב: `byte i = 0;`

ב. (2 נק׳) מחליפים את שורה 19 ב: `return x ^ 50;`

ג. (2 נק׳) מוחקים את שורה 24.

עבור כל מאורע, ציינו את **כל** שלבי הקומפילציה בהם הוא יכול להתרחש ונמקו בקצרה.

ד. (2 נק׳) השם acc מקושר לטיפוס כלשהו (מוזכר בשורות 23 ו-26).

ה. (2 נק׳) מחושב הסכום 16+32 (שורה 26).

חלק ב׳ : אופטימיזציות (15 נק׳)

נתונה התוכנית הבאה בשפת הביניים שנלמדה בכיתה, כאשר p1 הוא פרמטר ו-t0 הוא ערך חזרה (כלומר בשימוש בסוף התוכנית) :

```
1   t1 := p1
2   t2 := 3
3   t3 := t1 + t2
4   t4 := t1 * t2
5   t5 := 0
6   label1:
7   if t3 < t4 goto label2
8   if t5 > 9 goto label3
9   t5 := t5 + 1
10  t3 := t3 + t2
11  t3 := t3 + t2
12  goto label1
13  label2:
14  t6 := 1
15  goto label4
16  label3:
17  t6 := 0
18  label4:
19  t7 := t6 * t2
20  t0 := t6
```

א. (3 נק׳) בצעו את אלגוריתם החלוקה ל-basic blocks שנלמד בכיתה, וציירו את ה-CFG של התוכנית הנתונה. התשובה צריכה להיות בצורת תרשים.

ב. (12 נק׳) בצעו על התוכנית הנתונה את האופטימיזציות הבאות :

1. Constant propagation/folding (reaching defintions)
2. Copy propagation (aliasing analysis)
3. Useless code elimination (liveness analysis)

הראו את התוצאה של הרצת האנליזה הרלוונטית לכל אופטימיזציה, ואז את המצב הסופי של הקוד אחרי הפעלת כל האופטימיזציות. סמנו בכל שורה שבה התבצעה אופטימיזציה, סמנו איזו אופטימיזציה פעלה.

שאלה 2: אנליזה סטטית (30 נק׳)

ריק הוא גאון, ולכן הוא המציא שפת תכנות וסביבת זמן-ריצה שבה יש ניהול זיכרון אוטומטי, בהתבסס על שיטת Mark & Sweep. בשפה שלו יש משתנים לוקאליים וגלובליים, טיפוסי מצביעים בסגנון Java – כלומר מצביעים שיכולים להצביע אך ורק לאובייקטים, ותמיד לתחילת אובייקט (אין pointer arithmetic) – ומשפטים מהצורה הבאה:

```
A x;            // declare `x` as pointer to `A`
x := new A      // allocate object of class `A`
x := y          // copy pointer
x := y.f        // field read
x.f := y        // field write
```

ובנוסף ביטויים אריתמטיים ובוליאניים, משפטי תנאי ולולאות כמו בכל שפת תכנות סבירה והגיונית.

ריק מעוניין לדעת לגבי כל אובייקט שהוקצה דינמית בשורה כלשהי i בקוד שלו, האם הוא **יכול להיות נגיש (reachable)** לאחר שהפונקציה הנוכחית — זו שמכילה את השורה i — מסתיימת. אובייקט שנשאר נגיש לאחר שהפונקציה שהקצתה אותו הסתיימה נקרא אובייקט בורח (escaped). לדוגמה:

1 `void wubba() {` 2 `  Dub x;` 3 `  x := new Dub;` 4 `  x.dub();` 5 `  return;` 6 `}`	1 `Dub lubba() {` 2 `  Dub y;` 3 `  y := new Dub;` 4 `  y.dub();` 5 `  return y;` 6 `}`
`new Dub` at line 2: *not escaped*	`new Dub` at line 2: *escaped*

א. (15 נק׳) הניחו שבתכניות של ריק **אין שדות מטיפוס מצביע** בהגדרות של טיפוסים מורכבים וכן **אין מערכים**. הציעו לריק אנליזה **נאותה** – כזו שאם היא אומרת שהקצאה בשורה מסוימת אינה בורחת, אז האובייקט המוקצה בוודאות אינו בורח, באף ריצה אפשרית. מותר לאנליזה להיות שמרנית (כלומר, להגיד שאובייקט עלול לברוח מהפונקציה גם כאשר בכל הריצות הוא אינו נגיש למעשה לאחר היציאה מהפונקציה); אבל ריק דורש שתכתבו אנליזה מדויקת ככל האפשר, ולא כדאי להכעיס את ריק.

הגדירו את טווח הערכים (הדומיין), יחס הסדר, פעולת join, ופונקציות המעברים המתאימות עבור המשפטים השונים בשפה.

ב. (10 נק׳) הסבירו כיצד הייתם משנים את האנליזה אם בתכניות **ייתכנו שדות מטיפוס מצביע**, למשל

```
1   class Wubba { Lubba dub; }
      ⋮
```

101 `x = new Wubba;` 102 `x.y = new Lubba;` 103 `return x;` // lines 101,102 escaped	201 `x = new Wubba;` 202 `x.y = new Lubba;` 203 `return z;` // does line 202 escape? 🤔

בסעיף זה **יש לשמר את הנאותות, אך אין דרישה שהאנליזה תהיה מדויקת** (למעשה זה קשה מאד לדייק במקרים אלה, כמו בשורות 202, 203 בדוגמה). עם זאת, אין לפגוע בדיוק של האנליזה במקרים המתוארים בסעיף א.

הנחיה: נסו להשאיר את הדומיין כמות שהוא ולשנות רק את פונקציות המעברים (זה יכול לחסוך הרבה עבודה ותסכול).

ג. (5 נק׳) מורטי בוהה באנליזה שכתבתם, ותמה: ״אבל למה זה טוב בכלל לדעת אם אובייקט יכול או לא יכול לברוח מפונקציה?״

תארו למורטי **אופטימיזציה** שהקומפיילר של ריק יכול לבצע בעזרת האנליזה, ובאיזה מקרים היא מתאפשרת. התייחסו בתשובתכם לניהול זיכרון במחסנית ובערמה.

שאלה 3: ניתוח תחבירי וסמנטי (25 נק׳)

בתרגיל זה נבנה ניתוח תחבירי וסמנטי על מנת לחשב ביטויים חשבוניים בייצוג פרפיקסי (prefix).

בייצוג זה, ביטויים חשבוניים בנויים תחילה מהאופרטור ולאחר מכן משני האופרנדים (**מספרים בלבד**), למשל הביטוי הבא שערכו 8:

```
+ 3 5
```

נשים לב שייצוג פרפיקסי מיתר את הצורך בלסמן את סדר הפעולות בביטוי ע״י סוגריים. למשל הביטוי הבא בצורתו הרגילה, שערכו 18:

```
3 * ( 2 + 4 )
```

שקול לביטוי הבא בייצוג פרפיקסי:

```
* 3 + 2 4
```

בנוסף, נוכל לכתוב ביטויים מורכבים יותר, כגון הביטוי הבא בצורה הרגילה שערכו 6:

```
( 2 * ( 5 + 3 ) + 2 ) / 3
```

השקול לביטוי הבא בייצוג פרפיקסי:

```
/ + * 2 + 5 3 2 3
```

א. (5 נק׳) השלימו את כללי הגזירה על מנת להגדיר דקדוק חסר-הקשר המקבל את שפת הביטויים החשבוניים בייצוג פרפיקסי שנוכל לממשו באמצעות המנתח Bison שנלמד בקורס. הטרמינלים בדקדוק מסומנים בקו תחתון, ניתן להגדיר טרמינלים נוספים במידת הצורך.

- S → E
- E → _____
- E → _____
- OP → <u>+</u> | <u>-</u> | <u>*</u> | <u>/</u>

בסעיפים הבאים נבנה הגדרה מונחת תחביר (תכונות וכללים סמנטיים) שתדפיס את ערכו של ביטויי חשבוני בייצוג פרפיקסי הנגזר מהדקדוק שהגדרתם בסעיף א׳.

ב. (5 נק׳) הגדירו את **התכונות הסמנטיות** הדרושות על מנת להדפיס את ערכו של ביטויי חשבוני בייצוג פרפיקסי בעת הניתוח הסמנטי. הסבירו את תשובתכם.

ג. (15 נק׳) פרטו את **הכללים הסמנטיים** הדרושים באמצעות פסאודו-קוד בשימוש התכונות שהגדרתם בסעיף ב׳ על מנת להדפיס את ערכו של ביטויי חשבוני בייצוג פרפיקסי בעת הניתוח הסמנטי.

הנחיות:

- השתמשו **בתכונות נורשות בלבד** או **בתכונות נוצרות בלבד**.
- יש לבצע את הניתוח הסמנטי בזמן בניית עץ הגזירה.

שאלה 4: Backpatching (20 נק׳)

הוסיפו לשפת FanC מבנה בקרה עבור לולאת for מעל טווח עם פרדיקט. לולאה זו מכילה שתי רצפי פקודות, הרצף הראשון Statements1 ירוץ לכל איבר במידה והפרדיקט מחזיר עבורו true ו-Statements2 ירוץ במידה והפרדיקט יחזיר false. הטווח נתון ע״י NUM1 ו-NUM2 והינו כוללני ($[num1, num2]$). ID1 מכיל מזהה שמשמש כשם עבור האינדקס של הלולאה. ID2 מכיל מזהה של פונקציה שתשמש כפרדיקט. נתון כלל גזירה חדש לשפה:

$$Statement \rightarrow \underline{for}\ \underline{NUM_1}\ \underline{to}\ \underline{NUM_2}\ \underline{as}\ \underline{ID_1}\ \underline{with}\ \underline{ID_2}\ \underline{LBRACKET}\ Statements_1\ \underline{RBRACKET}$$

$$\underline{LBRACKET}\ Statements_2\ \underline{RBRACKET}$$

דוגמא:

```
bool p(int x) {
     return x < 2;
}
for 0 to 3 as x with p {
     printi(x);
     print("p(x) returned true");
} {
     printi(x);
     print("p(x) returned false");
}
```

תדפיס:

```
0
p(x) returned true
1
p(x) returned true
2
p(x) returned false
3
p(x) returned false
```

לצורך המימוש ניתן להניח שהתוכנה תמיד נכונה תחבירית וסמנטית, זאת אומרת שאין בעיות טיפוסים או דריסה של משתנים.

בנוסף, בדומה להרצאות, נוסף לשפת ביניים מבנה של קריאה לפונקציה. תחילה מצהירים על פרמטרים ולאחר מכן מפעילים את הפונקציה ומציינים את מספר הפרמטרים המועברים. לדוגמה קריאה לפונקציה f עם פרמטר אחד (t1):

```
param t1
```

a = call f, 1

א. (8 נק׳) הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ״ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות שאתם משתמשים בהן עבור כל משתנה.

ב. (12 נק׳) כתבו סכימת תרגום בשיטת backpatching המייצרת את פריסת הקוד שהצעתם בסעיף הקודם. על הסכימה להיות יעילה ככל האפשר, הן מבחינת זמן הריצה שלה והן מבחינת המקום בזיכרון שנדרש עבור התכונות הסמנטיות.

שימו לב:

- אין לשנות את הדקדוק ,למעט הוספת מרקרים M,N שנלמדו בכיתה בלבד.
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנה Statement יש כללי גזירה פרט לאלו המוצגים בשאלה.

סוף המבחן

נוסחאות ואלגוריתמים

כל ההגדרות מתייחסות לדקדוק $G = (V, T, P, S)$.

Top Down

$$\text{first}(\alpha) = \{\, t \in T \mid \alpha \Rightarrow^* t\beta \wedge \beta \in (V \cup T)^* \,\}$$

$$\text{follow}(A) = \{\, t \in T \cup \{\$\} \mid S\$ \Rightarrow^* \alpha A t \beta \wedge \alpha \in (V \cup T)^* \ \wedge \beta \in (V \cup T)^*(\varepsilon|\$) \,\}$$

$$\text{select}(A \to \alpha) = \begin{cases} \text{first}(\alpha) \cup \text{follow}(A) & \alpha \Rightarrow^* \varepsilon \\ \text{first}(\alpha) & \text{otherwise} \end{cases}$$

<u>הגדרה:</u> דקדוק G הוא LL(1) <u>אם ורק אם</u> לכל שני כללים ב- G השייכים לאותו משתנה A מתקיים:

$$\text{select}(A \to \alpha) \cap \text{select}(A \to \beta) = \varnothing$$

הגדרת טבלת המעברים $M : V \times (T \cup \{\$\}) \to P \cup \{\text{error}\}$ עבור דקדוק LL(1):

$$M[A, t] = \begin{cases} A \to \alpha & t \in \text{select}(A \to \alpha) \\ \text{error} & t \notin \text{select}(A \to \alpha) \text{ for all } A \to \alpha \in P \end{cases}$$

אלגוריתם מנתח LL(1):

```
Q.push(S)
while !Q.empty() do
     X = Q.pop()
     t = next token
     if X ∈ T then
          if X = t then MATCH
          else ERROR
     else // X ∈ V
          if M[X , t] = error then ERROR
          else PREDICT(X , t)
     end if
end while
t = next token
if t = $ then ACCEPT
else ERROR
```

Bottom Up

פריט LR(0) הוא $(A \to \alpha \bullet \beta)$ כאשר $A \to \alpha\beta \in P$

סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:

- בסיס: $closure(I) = I$.
- צעד: אם $(A \to \alpha \bullet B\beta) \in closure(I)$, אז לכל $B \to \gamma \in P$, גם $(B \to \bullet\gamma) \in closure(I)$

פונקציית המעברים של האוטומט:

$$\delta(I , X) = \bigcup \left\{ closure(A \to \alpha X \bullet \beta) \mid (A \to \alpha \bullet X\beta) \in I \right\}$$

פריט LR(1) הוא $(A \to \alpha \bullet \beta , t)$ כאשר $A \to \alpha\beta \in P$, $t \in T \cup \{\$\}$

סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:

- בסיס: $closure(I) = I$.
- צעד: אם $(A \to \alpha \bullet B\beta , t) \in closure(I)$, אז לכל $B \to \gamma \in P$ ולכל $x \in first(\beta t)$, גם $(B \to \bullet\gamma , x) \in closure(I)$

פונקציית המעברים של האוטומט:

$$\delta(I , X) = \bigcup \left\{ closure(A \to \alpha X \bullet \beta , t) \mid (A \to \alpha \bullet X\beta , t) \in I \right\}$$

הגדרת טבלת action למנתח SLR:

$$action[i , t] = \begin{cases} SHIFT_j & \delta(I_i ,t) = I_j \\ REDUCE_k & \text{rule k is } A \to \alpha,\ (A \to \alpha \bullet) \in I_i \text{ and } t \in follow(A) \\ ACCEPT & (S' \to S \bullet) \in I_i \text{ and } t = \$ \\ ERROR & \text{otherwise} \end{cases}$$

הגדרת טבלת action למנתח LR(1):

$$action[i , t] = \begin{cases} SHIFT_j & \delta(I_i ,t) = I_j \\ REDUCE_k & \text{rule k is } A \to \alpha \text{ and } (A \to \alpha \bullet , t) \in I_i \\ ACCEPT & (S' \to S \bullet , \$) \in I_i \text{ and } t = \$ \\ ERROR & \text{otherwise} \end{cases}$$

הגדרת טבלת goto למנתח SLR ו- LR(1):

$$goto[i , X] = \begin{cases} j & \delta(I_i , X) = I_j \\ error & \text{otherwise} \end{cases}$$

אלגוריתם מנתח shift/reduce:

```
Q.push(0)        // where 0 is the initial state of the prefix automaton
while true do
     k = Q.top().state
     t = next token
     do action[k , t]
end while
```

קוד ביניים

סוגי פקודות בשפת הביניים:

1. משפטי השמה עם פעולה בינארית
2. משפטי השמה עם פעולה אונרית
3. משפטי העתקה
4. קפיצה בלתי מותנה
5. קפיצה מותנה
6. הדפסה

```
x := y op z
x := op y
x := y
goto L
if x relop y goto L
print x
```

Data-Flow Analysis

ההגדרות מתייחסות ל – CFG: G=(V,E).

הצורה הכללית של המשוואות בחישוב סריקה קדמית:

$$in(B) = \bigcap_{(S,B)\in E} out(S) \quad \text{או} \quad in(B) = \bigcup_{(S,B)\in E} out(S)$$

$$out(B) = f_B\big(in(B)\big)$$

הצורה הכללית של המשוואות בחישוב סריקה אחורית:

$$out(B) = \bigcap_{(B,S)\in E} in(S) \quad \text{או} \quad out(B) = \bigcup_{(B,S)\in E} in(S)$$

$$in(B) = f_B\big(out(B)\big)$$

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
SET	set
AND	and
OR	or
NOT	not
TRUE	true
FALSE	false
RETURN	return
IF	if
ELSE	else
WHILE	while
BREAK	break
CONTINUE	continue
SC	;
COMMA	,
LPAREN	(
RPAREN	)
LBRACE	{
RBRACE	}
LBRACKET	[
RBRACKET	]
ASSIGN	=
RELOP	== | != | < | > | <= | >= | in
BINOP	+ | - | * | /
DOTS	..
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
STRING	"([^\n\r\"\\]|\\[rnt"\\])+"

דקדוק:

1. $Program \rightarrow Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
5. $RetType \rightarrow Type$
6. $RetType \rightarrow VOID$
7. $Formals \rightarrow \epsilon$
8. $Formals \rightarrow FormalsList$
9. $FormalsList \rightarrow FormalDecl$
10. $FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
11. $FormalDecl \rightarrow Type\ ID$
12. $FormalDecl \rightarrow EnumType\ ID$
13. $Statements \rightarrow Statement$
14. $Statements \rightarrow Statements\ Statement$
15. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
16. $Statement \rightarrow Type\ ID\ SC$
17. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow EnumType\ ID\ ASSIGN\ Exp\ SC$
19. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
20. $Statement \rightarrow Call\ SC$
21. $Statement \rightarrow RETURN\ SC$
22. $Statement \rightarrow RETURN\ Exp\ SC$
23. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
24. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
25. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
26. $Statement \rightarrow BREAK\ SC$
27. $Statement \rightarrow CONTINUE\ SC$
28. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
29. $Call \rightarrow ID\ LPAREN\ RPAREN$
30. $ExpList \rightarrow Exp$
31. $ExpList \rightarrow Exp\ COMMA\ ExpList$
32. $Type \rightarrow INT$
33. $Type \rightarrow BYTE$
34. $Type \rightarrow BOOL$
35. $Type \rightarrow SET\ LBRACKET\ NUM\ DOTS\ NUM\ RBRACKET$
36. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
37. $Exp \rightarrow Exp\ BINOP\ Exp$

38. $Exp \rightarrow ID$
39. $Exp \rightarrow Call$
40. $Exp \rightarrow NUM$
41. $Exp \rightarrow NUM\ B$
42. $Exp \rightarrow STRING$
43. $Exp \rightarrow TRUE$
44. $Exp \rightarrow FALSE$
45. $Exp \rightarrow NOT\ Exp$
46. $Exp \rightarrow Exp\ AND\ Exp$
47. $Exp \rightarrow Exp\ OR\ Exp$
48. $Exp \rightarrow Exp\ RELOP\ Exp$
49. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$