מבחן סוף סמסטר – מועד ב'

טור מקור

מרצה אחראי: ד"ר שחר יצחקי

מתרגלים: הילה לוי, תומר כהן, גיא ארבל, אנדריי בבין

הוראות:

1. בטופס המבחן 14 עמודים, מתוכם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
2. משך המבחן שלוש שעות (180 דקות).
3. כל חומר עזר חיצוני אסור לשימוש.
4. בשאלות הפתוחות, ניתן לציין לגבי סעיף או שאלה "לא יודע/ת" בלבד. תשובה זו תזכה ב-20% מהניקוד. תשובות שגויות לא יזכו בניקוד.
5. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.
6. אין צורך להגיש את טופס מבחן זה בתום הבחינה.
7. את התשובות לשאלות הסגורות יש לסמן בטופס התשובות הנפרד בלבד. את התשובות לשאלות הפתוחות יש לכתוב במקומות המסומנים בבחינה.
8. ודאו כי אתם מגישים טופס תשובות וטופס הבחינה בלבד (את מחברת הטיוטה ניתן לשמור אצלכם).

בהצלחה!

חלק א' - שאלות סגורות (45 נק')

שלבי קומפילציה (27 נק')

נתונה התוכנית הבאה בשפת FanC:

```
1.  int a = 15;
2.  int ƀ x = 1;
3.  int m = 1;
4.  while (a > 0) {
5.      a = a – 1;
6.      ƀ x = ƀ x * 2;
7.      m = (m * a) - ƀ x;
8.      {
9.          int c = 1;
10.         m = m * ƀ x;
11.     }
12. }
13. print("11");
```

בסעיפים הבאים (שאלות 1 עד 5) מוצגים שינויים (בלתי תלויים) לקוד של התוכנית. עבור כל שינוי ציינו את השלב המוקדם ביותר שבו נגלה את השגיאה:

שאלה מספר 1:

(3 נק') מחליפים את שורה 1 ב-

```
int a == 15;
```

א. שגיאה בניתוח תחבירי
ב. שגיאה בניתוח הסמנטי
ג. שגיאה בניתוח לקסיקלי
ד. אין שגיאה
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה מספר 2:

(3 נק') מחליפים את שורה 6 ב-

```
c = ƀ x * 2;
```

א. שגיאה בניתוח סמנטי
ב. שגיאה בניתוח תחבירי
ג. אין שגיאה
ד. שגיאה בניתוח לקסיקלי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה מספר 3:

(3 נק') מוחקים את שורה 8

א. שגיאה בניתוח תחבירי
ב. אין שגיאה
ג. שגיאה בניתוח לקסיקלי
ד. שגיאה בניתוח סמנטי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה מספר 4:

(3 נק') מחליפים את שורה 4 ב-
```
func(a > 0) {
```

א. שגיאה בניתוח סמנטי
ב. אין שגיאה
ג. שגיאה בניתוח לקסיקלי
ד. שגיאה בניתוח תחבירי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה מספר 5:

(3 נק') מחליפים את שורה 13 ב-
```
print(11);
```

א. שגיאה בניתוח סמנטי
ב. שגיאה בניתוח לקסיקלי
ג. שגיאה בניתוח תחבירי
ד. אין שגיאה
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שינויים ב-FanC- שאלות 6-7:

נרצה להוסיף לשפת FanC תמיכה בהוראה חדשה עבור קריאות לפונקציות ספרייה.
בכל הצהרה (statement) שהיא קריאה לפונקציה, ניתן להוסיף @ לפני שם הפונקציה כדי להפוך את הקריאה ל-inline.
המשמעות היא שבמקום לבצע פקודת CALL בזמן ריצה, גוף הפונקציה מודבק לתוך הקוד.

למשל, נניח כי קיימת פונקציית ספרייה בשם double_print אשר מקבלת כפרמטר מחרוזת בשם str והגוף שלה הוא הקוד הבא:

```
1. print(str);
2. print(str);
```

הקומפיילר של FanC יפלוט קוד זהה עבור שתי התוכניות הבאות:

```
1. @double_print("A");
2. if (11 > 22) {
3.       double_print("B");
4. }
```

```
1. print("A");
2. print("A")
3. if (11 > 22) {
4.    double_print("B");
5. }
```

שאלה מספר 6:

(6 נק') על מנת לתמוך בפיצ'ר החדש, מהו שלב הקומפילציה המוקדם ביותר מבין הרשומים כאן בו **לא** נבצע שינוי?

א. נצטרך לערוך שינויים בכל אחד משלבי הקומפילציה שרשומים כאן.
ב. ניתוח לקסיקלי
ג. ייצור קוד
ד. ניתוח תחבירי

שאלה מספר 7:

(6 נק') בזמן שעבדת על מימוש הפיצ'ר החדש, השותפה שלך לתרגילי הבית הוסיפה ל-FanC תמיכה ב-JIT.
על מנת לבדוק את יעילות הפיצ'ר החדש, הוספת דגל "no-inline--" אשר מכבה את פעולת ה-inline (מתעלם מהתו @).
כאשר משווים ריצה בלי inline לריצה עם inline (בשתי הריצות יש JIT), נסתכל על הטענות הבאות:

1. עבור כל תוכנית inline מגדיל את קוד הביניים שנוצר.
2. עבור כל תוכנית inline מקטין או לא משנה את זמן הריצה.

איזו מהטענות נכונה?

א. אף טענה אינה נכונה.
ב. רק הטענה הראשונה נכונה.
ג. רק הטענה השנייה נכונה.
ד. שתי הטענות נכונות.

אופטימיזציות (18 נק')

שאלה מספר 8:
(6 נק') ליוסי נתון הקוד הבא:

```
1.  x = y + 1
2.  if (x > 10) goto 5
3.  z = 0
4.  goto 7
5.  z = x - 10
6.  goto 7
7.  y = z * x
8.  a = y * y
```

יוסי לא אוהב הרבה בלוקים בסיסיים, הוא הריץ את האלגוריתם שלמד בכיתה על מנת למצוא את מספר הבלוקים המינימלי של קטע הקוד שלו.
יוסי הופתע לרעה מכמות הבלוקים ולכן הוא מוכן לסכן את נכונות הקוד שלו על ידי מחיקת שורה בודדת!
כאשר מוחקים שורה נשים במקומה את הפקודה NOPE אשר אינה עושה דבר.
מהו השינוי הגדול ביותר שיוסי יכול לעשות למספר הבלוקים הבסיסיים המינימלי על ידי מחיקה של שורה בודדת?

א. קטן ב- 1
ב. קטן ב- 2
ג. קטן ב- 3
ד. מספר הבלוקים המינימלי לא משתנה לא משנה איזו שורה נמחק.

שאלות 9-10:

שאלה מספר 9:

נתון קטע הקוד הבא:

```
x = 4
n = x + 1
m = y + n
label0: z = y * x
x = x - 1
if x > 0 goto label0
z = y + 5
```

(6 נק') ציירו את ה-CFG של קוד הביניים הנתון. מה מספר הצמתים והקשתות בגרף?

א. 3 קשתות, 3 צמתים
ב. 3 קשתות, 4 צמתים
ג. 4 קשתות, 3 צמתים
ד. 4 קשתות, 4 צמתים
ה. אף תשובה אינה נכונה

שאלה מספר 10:

(6 נק') על קוד הביניים הכולל שמכיל את קטע זה הופעלה תחילה אנליזת חיות אשר דיווחה כי לאחר קטע קוד זה חיים המשתנים m ו-z בלבד.

כעת עליך להריץ את כל האופטימיזציות השונות שנלמדו בכיתה עד לקבלת קוד אופטימלי.

איזו אופטימיזציה **אינה** מתבצעת?

א. כל האופטימיזציות שמצויינות בתשובות מתבצעות.
ב. Commonsub-expression elimination
ג. Constant propagation
ד. Constant folding
ה. Useless code elimination

חלק ב' - שאלות פתוחות (55 נק')

שאלה 1: דקדוקים (25 נק')

נתון הדקדוק הבא:

$S \rightarrow A = B$
$A \rightarrow id$
$B \rightarrow B\ OP\ num$
$B \rightarrow num$
$OP \rightarrow * \mid +$

משתנים מסומנים באותיות אנגליות גדולות ואסימונים באותיות אנגליות קטנות.
כאשר האסימונים =,+,* הם אסימונים אשר מבצעים את פעולות החשבון הרגילות והאסימונים id ו-num מייצגים את הביטויים הרגולרים המתאימים לאסימונים אלה בשפת FanC מתרגילי הבית שלכם.

א. (10 נק') הראו שהדקדוק שייך לSLR, בתשובה שלכם הראו את האוטומט, Follows וטבלת action-goto.

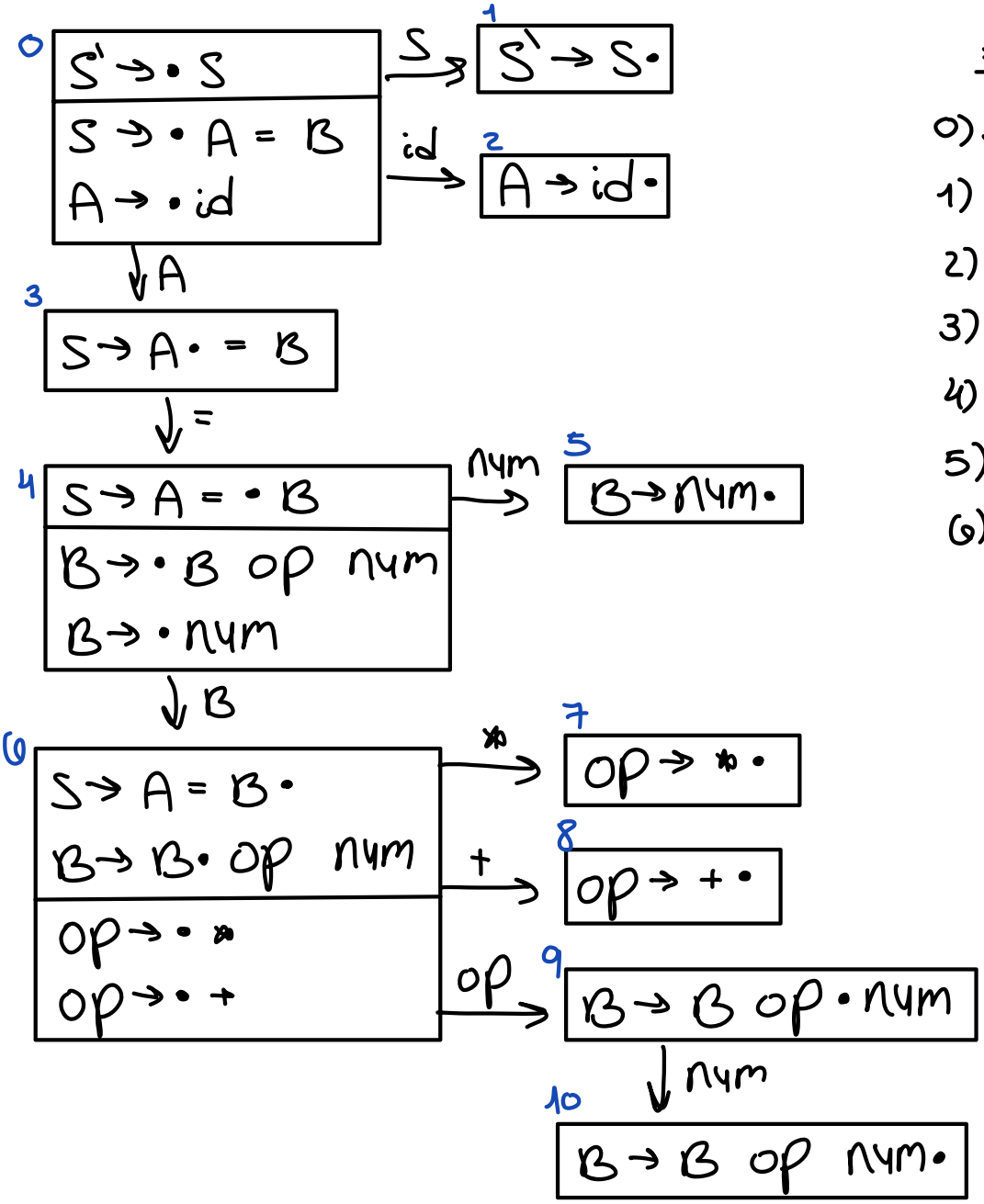
ב. (10 נק') נתחו את הקלט הבא:

$$x = 2 + 3 * 6$$

הציגו את כל מצבי המחסנית במהלך הניתוח.

ג. (5 נק') בהנחה שמשלימים את מימוש הקומפיילר ומריצים את התוכנית, מהו ערכו של x?

פתרון לשאלת דקדוקים:



מספור הכללים:

0) S' → S
1) S → A = B
2) A → id
3) B → B op num
4) B → num
5) op → *
6) op → +

Follows:

Follow(S') = {$}
Follow(S) = {$}
Follow(A) = {=}

Follow(B) = {*, +, $}
Follow(op) = {num}

טבלת action-goto:

States	action						goto			
	=	id	num	*	+	$	S	A	B	OP
0		s2					1	3		
1						Acc				
2	r2									
3	s4									
4			s5						6	
5				r4	r4	r4				
6				s7	s8	r1				9
7			r5							
8			r6							
9			s10							
10				r3	r3	r3				

ב. ניתוח המילה:

מחסנית	פעולה	קלט
(0,)	s2	id = num + num * num
(0,) (2, id)	r2	= num + num * num
(0,) (3, A)	s4	= num + num * num
(0,) (3, A) (4, =)	s5	num + num * num
(0,) (3, A) (4, =) (5, num)	r4	+ num * num
(0,) (3, A) (4, =) (6, B)	s8	+ num * num
(0,) (3, A) (4, =) (6, B) (8, +)	r6	num * num
(0,) (3, A) (4, =) (6, B) (9, OP)	s10	num * num
(0,) (3, A) (4, =) (6, B) (9, OP) (10, num)	r3	* num
(0,) (3, A) (4, =) (6, B)	s7	* num
(0,) (3, A) (4, =) (6, B) (7, *)	r5	num

(0,) (3, A) (4, =) (6, B) (9, OP)	s10	num
(0,) (3, A) (4, =) (6, B) (9, OP) (10, num)	r3	$
(0,) (3, A) (4, =) (6, B)	r1	$
(0,) (1, S)	acc	$

ג. <u>ערכו של x הוא:</u> **30**

שאלה 2: אנליזה סטטית (30 נק')

טונטו חזר משירות מילואים ונזכר שלא סיים את מימוש הקומפיילר שעליו הוא עבד בסמסטר שעבר. הוא עצר בשלב שבו יש לו frontend אשר הפלט שלו הוא קוד ביניים בצורה של 3AC. הנה כמה דוגמאות של תוכניות שנוצרו, מחולקות לבלוקים בסיסיים.

```
absur
0:      w0 := readi()
        if w0 > 0 goto 2
1:      w1 := –w0
2:      t0 := readi()
        t1 := 3 * t0
        goto 3
3:      w2 := φ(w0, w1)
        t2 := w2 + t1
        printi(t2)
```

```
signur
0:      w0 := readi()
        if w0 > 0 goto 5
1:      if w0 = 0 goto 3
2:      x0 := 45
        goto 4
3:      x1 := 48
4:      print("nisse")
        goto 6
5:      x2 := 43
6:      x4 := φ(x0, x1, x2)
        printi(x4)
```

```
sum_squr
0:      n := readi()
        i0 := 1
        sum0 := 0
1:      i1 := φ(i0, i2)
        if i1 >= n goto 4
2:      t0 := i1 * i1
        sum1 := φ(sum0, sum2)
        sum2 := sum1 + t0
        goto 3
3:      i2 := i1 + 1
        goto 1
4:      sum4 := φ(sum0, sum2)
        printi(sum4)
```

כעת הוא חייב להשלים בהקדם את ה-backend ולייצר קוד אסמבלי. לצורך כך הוא מעוניין להמיר את קוד הביניים שלו לתצורה של LLVM IR כדי שיוכל להשתמש בספריה של LLVM וכך לסיים את המשימה מהר יותר. טונטו נזכר מה הייתה הבעיה המרכזית שבגינה לא הספיק להשלים את המשימה לפני תקופת מועדי א׳: בתשתית LLVM, האופרטור φ (פִּי) צריך לקבל, בנוסף למשתנים המייצגים את הערכים, גם את הבלוק שממנו מגיעה ההגדרה של כל אחד מהארגומנטים.

יתר על כן, כל ציון בלוק המוצמד לארגומנט חייב להיות **בלוק קודם ישיר** (direct predecessor) של הבלוק שבו מופיע השימוש באופרטור φ — כלומר חייבת להיות **קשת אחת בדיוק** המקשרת ביניהם, ואסורים מסלולים באורך גדול מ-1.

בתור שלב ראשון, טונטו מריץ על הקוד אנליזה בשם Elvish Definitions, שהיא גרסה של Reaching Definitions שהותאמה על-ידי אלפור לקוד SSA באופן הבא:

$L = P(Vars)$; $\sqsubseteq\ =\ \subseteq$

Statement at ℓ	kill(ℓ)	gen(ℓ)
$v := \varphi(u_1, \ldots, u_k)$	$\{ u_1, \ldots, u_k \}$	$\{ v \}$
$v := expr$ (that is not φ)	$\varnothing$	$\{ v \}$
anything else	$\varnothing$	$\varnothing$

שימו לב: מכיוון שב-SSA לכל משתנה יש בדיוק הגדרה אחת, אלפור החליט שמספיק לשמור רק את שמות המשתנים באנליזה (Vars). כמו כן הוסכם כי הגדרה של משתנה מפסיקה להיות תקפה ברגע שהמשתנה משמש כארגומנט לאופרטור φ.

א. (10 נק') ציירו את גרף בקרת הזרימה של כל אחת מהתוכניות שבטבלה בראש העמוד, והריצו את אנליזת Elvish Definitions על הגרפים שהתקבלו. יש לרשום תוצאות סופיות בלבד (in ו-out של כל בלוק).

(הבלוקים באנליזה הם הבלוקים הבסיסיים של התוכנית בשפת הביניים; כאשר בבלוק יש מספר השמות ברצף, מופעלות, כרגיל, פונקציות המעבר ברצף בזו אחר זו.)

ב. (20 נק') כעת על טונטו להחליף את הביטויים מסוג $\varphi(u_1, \ldots, u_k)$ לביטויים מסוג שמתאים ל LLVM IR, כלומר מהצורה הבאה:

$v := \varphi([u_1, \ell_1], \ldots, [u_k, \ell_k])$

שבו כל זוג מורכב משם של משתנה u_i ומתווית של בלוק ℓ_i, הקודם לבלוק שבו נמצאת ההשמה, כך שבכל פעם שהריצה מגיעה אל ההשמה מן הבלוק ℓ_i, המשתנה v יקבל את הערך של u_i.

טונטו שם לב, כי ההגדרה של u_i לא בהכרח מגיעה מבלוק קודם ישיר, ולכן לא ניתן להשתמש בתווית הזאת. למשל: בתוכנית absur, הוא **לא יכול** להחליף את ההשמה בבלוק 3 באופן הבא:

$w2 := \varphi([w0, 0], [w1, 1])$ ← הביטוי הזה **אינו חוקי** (בבלוק 3)

יתרה מכך, המסלול שמוביל מהגדרות של u_i שונים עשוי לעבור דרך **אותו** בלוק ישיר קודם. למשל: בתוכנית absur, ההגדרות של w0, w1 מגיעות שתיהן לבלוק 3 דרך בלוק 2 (שהוא הבלוק הקודם היחיד של 3). כלומר, טונטו **גם לא יכול** להחליף את ההשמה בבלוק 3 באופן הבא:

$w2 := \varphi([w0, 2], [w1, 2])$ ← הביטוי הזה **אינו חוקי**

הפתרון היחיד במקרה זה הוא להוסיף משתנה זמני חדש בתחילת בלוק 2:

$t\_w0\_w1 := \varphi([w0, 0], [w1, 1])$

ולשנות את ההשמה בתחילת בלוק 3 ל-

$w2 := t\_w0\_w1$

העיקרון המנחה הוא שכל פקודת φ תופיע במיקום **הראשון** שבו ההגדרות (של w0 ו-w1, במקרה זה) **נפגשות**.

עזרו לטונטו לכתוב אנליזה סטטית שבעזרתה יוכל הקומפיילר שלו להוסיף את פקודות φ החדשות וכן לשנות את כל פקודות φ הקיימות בהתאם. טונטו גילה בפתקית נייר שנשכחה בין כריות של ספה ישנה במעונות כי הדומיין של האנליזה עשוי להיות

$L = P(Lab \times P(Vars))$; $\sqsubseteq\ = \subseteq$

כאשר Vars = קבוצת המשתנים, Lab = קבוצת התוויות.

הגדירו את פונקציות המעברים. הקפידו על הגדרות מתמטיות מדויקות, תוך שימוש בסימונים הבאים:

קבוצת הבלוקים (תוויות) שהם הקודמים הישירים של הבלוק בעל התווית ℓ	$pred(\ell)$
התוצאות של האנליזה Elvish Definitions, על הבלוק ℓ	$Ein(\ell), Eout(\ell)$
קבוצה של קבוצות של משתנים המתארת את כל המופעים של φ (בתוכנית המקורית); כל איבר ב-C הוא קבוצה של משתנים המופיעים יחד כאופרנדים בפקודת φ בודדת כלשהי	C

מותר להניח כי:

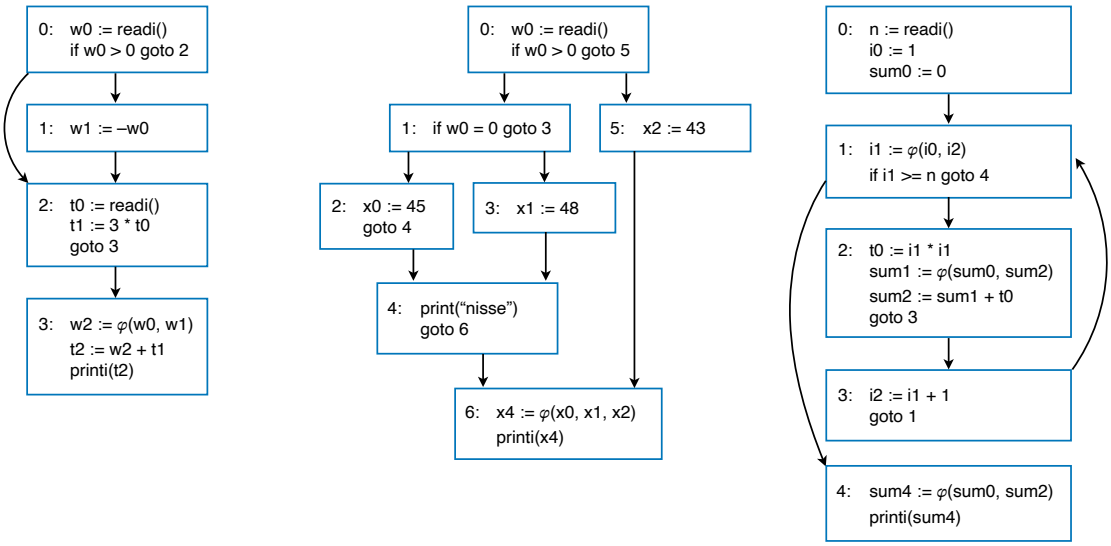
- התוכנית היא תקינה ובצורת SSA.
- כל השימושים במשתנים בתוכנית מכבדים את ההגדרה של Elvish Definitions, כלומר יש הגדרה של המשתנה שמגיעה לאותו שימוש לפי האנליזה של אלפור.
- דרגת הכניסה של כל בלוק בסיסי בגרף בקרת הזרימה היא **לכל היותר 2**.

הריצו את האנליזה על שלושת הגרפים מסעיף א. יש לרשום תוצאות סופיות בלבד (in ו-out של כל בלוק).

הסבירו כיצד הקומפיילר ישתמש בתוצאות האנליזה על מנת להמיר את התוכנית לצורה הרצויה. הדגימו זאת על התוכניות signur ו-sum_squr.

פתרון

סעיף א



absur

ℓ	in(ℓ)	out(ℓ)
0	∅	{ w0 }
1	{ w0 }	{ w0, w1 }
2	{ w0, w1 }	{ w0, w1, t0, t1 }
3	{ w0, w1, t0, t1 }	{ t0, t1, w2, t2 }

signur

ℓ	in(ℓ)	out(ℓ)
0	∅	{ w0 }
1	{ w0 }	{ w0 }
2	{ w0 }	{ w0, x0 }
3	{ w0 }	{ w0, x1 }
4	{ w0, x0, x1 }	{ w0, x0, x1 }
5	{ w0 }	{ w0, x2 }
6	{ w0, x0, x1, x2 }	{ w0, x4 }

sum_squr

ℓ	in(ℓ)	out(ℓ)
0	∅	{ n, i0, sum0 }
1	{ n, i0, i1, i2, sum0, sum1, sum2, t0 }	{ n, i1, sum0, sum1, sum2, t0 }
2	{ n, i1, sum0, sum1, sum2, t0 }	{ n, i1, sum1, sum2, t0 }
3	{ n, i1, sum1, sum2, t0 }	{ n, i1, i2, sum1, sum2, t0 }
4	{ n, i1, sum0, sum1, sum2, t0 }	{ n, i1, sum0, sum1, sum2, sum4, t0 }

<u>סעיף ב</u>

פונקציית המעברים: אם $\text{pred}(\ell) = \{u_1, u_2\}$ עבור $u_1 \neq u_2$ (כלומר, דרגת הכניסה של ℓ היא 2):

$$\text{out}(\ell) = \text{in}(\ell) \cup \left\{(\ell, c') \mid c \in C \wedge c' = c \cap \text{Ein}(\ell) \wedge |c'| > 1 \wedge \left(c' \not\subseteq \text{Eout}(u_1) \vee c' \not\subseteq \text{Eout}(u_2)\right)\right\}$$

אחרת (דרגת הכניסה היא 1):

$$\text{out}(\ell) = \text{in}(\ell)$$

הסבר: בתיאור המילולי הוסבר כי האנליזה צריכה לזהות את המקום בו שתי הגדרות, השייכות לאיזו שהיא פקודת φ בתוכנית, נפגשות. שני מסלולים בגרף יכולים להיפגש רק בצומת שדרגת הכניסה שלו גדולה מ-1, ולפי ההנחיות זה חייב להיות 2. המשמעות של הגדרות "נפגשות" היא שכולן מופיעות ב-ℓ ולעומת זאת בבלוקים הקודמים ל-ℓ (לפחות באחד מהם) לא כולן הופיעו. למשל: בכניסה לבלוק 2 של התוכנית absur, כבר נמצאות ההגדרות של w0, w1. אבל בבלוק 0 הקודם לבלוק 2 חסרה ההגדרה של w1. לכן התנאי שלמעלה $\left(c' \not\subseteq \text{Eout}(u_1) \vee c' \not\subseteq \text{Eout}(u_2)\right)$ מתקיים, והקבוצה {w0, w1} תתווסף ל-out(2) באנליזה.

כיצד הקומפיילר ישתמש בתוצאה:

בכל בלוק ℓ, אם ב-out(ℓ) קיים זוג $(\ell, \{v_1, v_2\})$, אז נוסיף בהתחלה של הבלוק פקודת φ מהצורה::

$$t\_v_1\_v_2 := \varphi([v_1, u_1], [v_2, u_2])$$

כאשר $\{u_1, u_2\} = \text{pred}(\ell)$, והשיוך שלהם ל v_1, v_2 הוא כך ש $v_1 \in \text{Eout}(u_1)$ ו $v_2 \in \text{Eout}(u_2)$.

מקרה מיוחד הוא כשקיים זוג (ℓ, c) כאשר $|c| > 2$. כך למשל בתוכנית signur יש הגדרה בבלוק 6 עם הקבוצה {x0, x1, x2}. במקרה זה **חייבת** להיות תת-קבוצה $c' \subset c$ ותווית נוספת ℓ' כך ש $(\ell', c') \in \text{in}(\ell)$. הזוג הזה מייצג פקודת φ קודמת; בדוגמה זו $(4, \{x0, x1\}) \in \text{in}(6)$ ולפיכך ההגדרה שתתווסף בתחילת בלוק 6 תהיה:

$$t\_x0\_x1\_x2 := \varphi([t\_x0\_x1, 4], [x2, 5])$$

המשתנה x2 הוא זה שנותר: $c \setminus c' = \{x2\}$. גם פה, אם $|c \setminus c'| > 1$, כי אז **חייב** להיות זוג **נוסף** $(\ell'', c \setminus c') \in \text{in}(\ell)$ והפעולה תתבצע באופן זהה עבור הארגומנט השני.

בחירת התוויות תיעשה על פי אותה חוקיות: $\{u_1, u_2\} = \text{pred}(\ell)$, וכן $c' \subseteq \text{Eout}(u_1)$ ו $c \setminus c' \subseteq \text{Eout}(u_2)$.

לבסוף, כל $\varphi(v_1,\ldots,v_k)$ שהופיע בתוכנית המקורית מוחלף במשתנה העזר המתאים $t\_v_1\_\ldots\_v_k$.

absur

ℓ	in(ℓ)	out(ℓ)
0	∅	∅
1	∅	∅
2	∅	{ (2, {w0,w1}) }
3	{ (2, {w0,w1}) }	{ (2, {w0,w1}) }

signur

ℓ	in(ℓ)	out(ℓ)
0	∅	∅
1	∅	∅
2	∅	∅
3	∅	∅
4	∅	{ (4, {x0, x1}) }
5	∅	∅
6	{ (4, {x0, x1}) }	{ (4, {x0, x1}), (6, {x0, x1, x2}) }

sum_squr

ℓ	in(ℓ)	out(ℓ)
0	∅	∅
1	{ (1, {i0, i2}), (1, {sum0, sum2}) }	{ (1, {i0, i2}), (1, {sum0, sum2}) }
2	{ (1, {i0, i2}), (1, {sum0, sum2}) }	{ (1, {i0, i2}), (1, {sum0, sum2}) }
3	{ (1, {i0, i2}), (1, {sum0, sum2}) }	{ (1, {i0, i2}), (1, {sum0, sum2}) }
4	{ (1, {i0, i2}), (1, {sum0, sum2}) }	{ (1, {i0, i2}), (1, {sum0, sum2}) }

התוכניות לאחר השינויים כפי שהוסבר (השורות שהשתנו מודגשות בצבע):

signur

```
0:      w0 := readi()
        if w0 > 0 goto 5
1:      if w0 = 0 goto 3
2:      x0 := 45
        goto 4
3:      x1 := 48
4:      t_x0_x1 := φ([x0,2], [x1,3])
        print("nisse")
        goto 6
5:      x2 := 43
6:      t_x0_x1_x2 := φ([t_x0_x1, 4], [x2, 5])
        x4 := t_x0_x1_x2
        printi(x4)
```

sum_squr

```
0:      n := readi()
        i0 := 1
        sum0 := 0
1:      t_i0_i2 := φ([i0, 0], [i2, 3])
        t_sum0_sum2 := φ(sum0, sum2)
        i1 := t_i0_i2
        if i1 >= n goto 4
2:      t0 := i1 * i1
        sum1 := t_sum0_sum2
        sum2 := sum1 + t0
        goto 3
3:      i2 := i1 + 1
        goto 1
4:      sum4 := t_sum0_sum2
        printi(sum4)
```

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<u>הגדרה:</u> דקדוק G הוא LL(1) <u>אם ורק אם</u> לכל שני כללים ב- G השייכים לאותו משתנה A מתקיים:
$select(A \to \alpha) \cap select(A \to \beta) = \varnothing$

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

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

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 `x := y op z`
2. משפטי השמה עם פעולה אונרית `x := op y`
3. משפטי העתקה `x := y`
4. קפיצה בלתי מותנה `goto L`
5. קפיצה מותנה `if x relop y goto L`
6. הדפסה `print x`

Data-Flow Analysis

ההגדרות מתייחסות ל-CFG מהצורה $G = (V, E)$:

הצורה הכללית של המשוואות בחישוב סריקה קדמית:

$$\mathrm{in}(B) = \bigsqcup_{(S,B)\in E} \mathrm{out}(S)$$
$$\mathrm{out}(B) = f_B\big(\mathrm{in}(B)\big)$$

הצורה הכללית של המשוואות בחישוב סריקה אחורית:

$$\mathrm{in}(B) = \bigsqcup_{(B,D)\in E} \mathrm{out}(D)$$
$$\mathrm{out}(B) = f_B\big(\mathrm{in}(B)\big)$$

שפת FanC

אסימונים:

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

דקדוק:

1. $Program \rightarrow Statements$
2. $Statements \rightarrow Statement$
3. $Statements \rightarrow Statements\ Statement$
4. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
5. $Statement \rightarrow Type\ ID\ SC$
6. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
7. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
8. $Statement \rightarrow Call\ SC$
9. $Statement \rightarrow RETURN\ SC$
10. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
11. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
12. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
13. $Statement \rightarrow BREAK\ SC$
14. $Statement \rightarrow CONTINUE\ SC$
15. $Call \rightarrow ID\ LPAREN\ Exp\ RPAREN$
16. $Type \rightarrow INT$
17. $Type \rightarrow BYTE$
18. $Type \rightarrow BOOL$
19. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
20. $Exp \rightarrow Exp\ BINOP\ Exp$
21. $Exp \rightarrow ID$
22. $Exp \rightarrow Call$
23. $Exp \rightarrow NUM$
24. $Exp \rightarrow NUM\ B$
25. $Exp \rightarrow STRING$
26. $Exp \rightarrow TRUE$
27. $Exp \rightarrow FALSE$
28. $Exp \rightarrow NOT\ Exp$
29. $Exp \rightarrow Exp\ AND\ Exp$
30. $Exp \rightarrow Exp\ OR\ Exp$
31. $Exp \rightarrow Exp\ RELOP\ Exp$
32. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$