מבחן סוף סמסטר – מועד א'

פתרון

מרצה אחראית: ד"ר שחר יצחקי

מתרגלים: מתן פלד, רון אלעד, תומר כהן, הילה לוי

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". 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 במחברת הבחינה.
8. ודאו כי אתם מגישים טופס תשובות ומחברת בחינה בלבד.

בהצלחה!

חלק א' - שאלות סגורות (50 נק')

שלבי קומפילציה (20 נק')

נתונה התוכנית הבאה בשפת FanC:

```
1  byte fix(byte a, byte div) {
2      return a/div;
3  }
4
5  byte foo(byte a, byte b) {
6      byte res = a * b;
7      if (res < a) {
8          return foo(fix(a, res), b);
9      }
10     return res;
11 }
12
13 void main() {
14     byte a = 4b;
15     byte b = 2b;
16     foo(a, b);
17 }
```

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 1 (2 נק')

מחליפים את האסימון הראשון בשורה 14 באסימון `auto`.

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

שאלה 2 (2 נק')

מחליפים את טיפוס החזרה של `fix` ל-`bool`.

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

שאלה 3 (2 נק')

משנים את שורה 2 ל- "`return a%div;`".

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

שאלה 4 (2 נק')

מחליפים את שורה 8 ב- "`return a/(div-div);`".

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

שאלה 5 (2 נק')

משנים את שורה 9 ל- "`return (int)res;`".

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

נרצה להוסיף לשפת FanC תמיכה ב- _ (קו תחתון) בדומה לאופן בו משתמשים ב-python interpreter.
לכל פונק' נשמור את הערך חזרה של קריאה לפונק' שמחזירה לא void וגם לא בוצעה השמה של הערך לתוך משתנה. לדוגמה, בקוד הבא, המשתנה a יחזיק את הערך 5:

```
int bar() {
    return 5;
}
void main() {
    bar();
    int a = _;
}
```

שאלה 6 (5 נק')

בשביל התמיכה בפיצ'ר החדש, באילו שלבי קומפילציה מבין הרשומים כאן, לא נבצע שינויים?

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

שאלה 7 (5 נק')

בעת מימוש תמיכה בפיצ'ר החדש עבור FanC, בשלב ייצור הקוד, האם יש להשתמש ב-backpatching כמו שלמדנו בכיתה?

א. לא, השינויים בשלב ייצור הקוד לא דורשים ביצוע שלbackpatching
ב. כן, אבל לא נצטרך לעשות emit לקוד נוסף, אפשר לפתור רק באמצעות קריאות ל-backpatch.
ג. לא, כי לא נעשה שינויים בשלב ייצור הקוד.
ד. כן, ונצטרך לקרוא גם ל-emit וגם ל-backpatch במימוש.
ה. כן, אבל רק אם מאתחלים משתנה חדש בכותרת של הלולאה.

אופטימיזציות (10 נק')

שאלה 8 (5 נק')

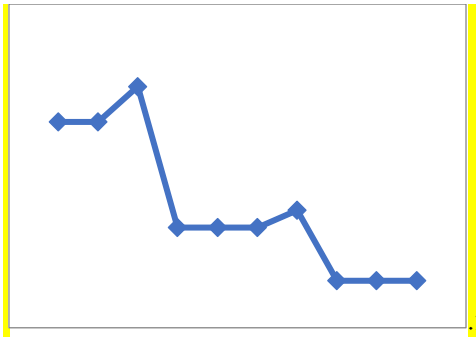
מהנדס בכיר רצה לבדוק את זמני הריצה של הקוד FanC שלו. לצורך כך, הוא בנה מעין benchmark שמריץ את הקוד 10 פעמים בלולאה ומודד את זמן הריצה של כל איטרציה של הלולאה. הקוד של ה-benchmark נראה כך:

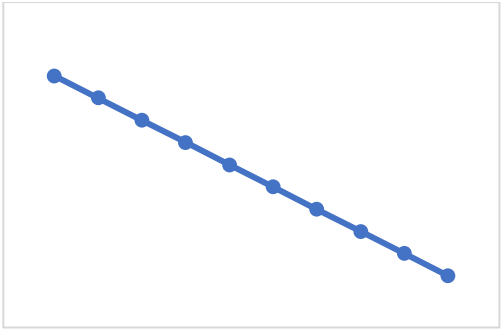
```
1   void main() {
2       int i = 0;
3       int time = gettime();
4       while (i < 10) {
5           code_under_test();
6           printi(gettime() - time);
7           time = gettime();
8           i = i + 1;
9       }
10  }
```

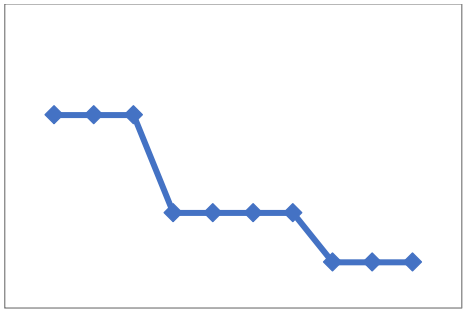
כאשר gettime היא פונקציה שמחזירה את הזמן הנוכחי ביחידת מידה שימושית, ו-code_under_test היא הפונקציה שאת הביצועים שלה אנחנו מעוניינים למדוד.

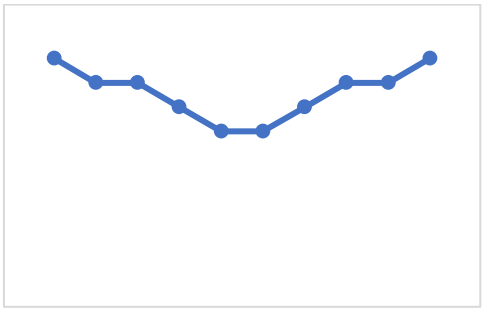
הקוד קומפל ל-LLVM IR באמצעות הקומפיילר של FanC כפי שמימשתם אותו בתרגיל בית 5, וכמו כן הורץ באמצעות lli עם כל האופטימיזציות שלמדנו בקורס (כולל JIT). ניתן להניח שאחרי כל איטרציה מערכת ההפעלה עושה context switch והמעבד מפנה את זיכרון המטמון, כלומר שאפשר להתעלם מאופטימזציות ברמת מערכת ההפעלה והמעבד.

בחרו את הגרף שמייצג ריצה אפשרית של ה-benchmark המתאימה לעקרונות שלמדנו בכיתה. בכל גרף ציר ה-x הוא מספר האיטרציה, וציר ה-y הוא המספר שהודפס עבור האיטרציה הזו, כלומר הזמן שהאיטרציה לקחה:


א.


ג.


ב.


ד.

ה. אף תשובה אינה נכונה.

שאלה 9 (5 נק')

אחרי שמימשנו את הפיצ'ר החדש מהשאלות הקודמות (_), הוספנו לקומפיילר גם שתי אופטימזציות חדשות:

- function inlining, שמחליפה פקודות call בעותק של תוכן הפונקציה הנקראת

- type narrowing, שמבצעת אנליזת abstract interpretation ומחזירה אינטרוולים עבור כל המשתנים המספריים. אם לפי תוצאות האנליזה טווח הערכים שמשתנה מסוג int מכיל מוגבל לטווח ש-byte יכול לייצג, האופטימיזציה תשנה אותו ל-byte.

האופטימיזציות מומשו לפי העקרונות שלמדנו בכיתה והן מתכנסות.

לאחר הפעלת כל האופטימיזציות שלמדנו (כולל החדשות), מה ניתן להגיד בוודאות על הקוד שהתקבל לעומת הקוד לפני אופטימיזציה?

א. אף תשובה אינה נכונה.
ב. הקוד לאחר אופטימיזציה קטן יותר (פחות שורות קוד).
ג. הקוד לאחר אופטימיזציה משתמש בפחות זיכרון במהלך זמן הריצה.
ד. זמן הריצה של הקוד לאחר אופטימיזציה מהיר יותר.

דקדוקים (20 נק')

נגדיר את הדקדוק G_1:

$$S \to \underline{(}\ S\ \underline{)}\ S$$
$$S \to \varepsilon$$

שאלה 10 (4 נק')

מהו מספר המצבים בפיתוח אוטומט LR(1) כפי שנלמד בתרגול עבור G_1?

א. 10
ב. 12
ג. 14
ד. אף תשובה אינה נכונה.

שאלה 11 (4 נק')

מהי דרגת היציאה המקסימלית באוטומט?

א. 2
ב. 3
ג. 4
ד. אף תשובה אינה נכונה.

נגדיר את הדקדוק G_2:

$$S \to E$$
$$E \to E\ \underline{*}\ B$$
$$E \to E\ \underline{+}\ B$$
$$E \to B$$
$$B \to 0$$
$$B \to 1$$

שאלה 12 (4 נק')

נסמן ב-A את מספר המצבים המתקבלים מפיתוח אוטומט LR(0) עבור הדקדוק G_2,
ונסמן ב-C את מספר המצבים המתקבלים מפיתוח אוטומט LR(1) עבור הדקדוק G_2.

א. A=10, C=10
ב. A=9, C=10
ג. A=10, C=8
ד. A=9, C=8

שאלה 13 (4 נק')

האם המצב הבא ייתכן בפיתוח מנתח LR(1) של הדקדוק G_2?

$$S \to E \cdot$$
$$E \to E \cdot * B$$

א. לא, אלה פריטי LR(0) ולא קיים מצב כזה בפיתוח LR(0) עבור הדקדוק G_2
ב. כן
ג. לא, אלה פריטי LR(1) אבל אין מצב כזה בפיתוח
ד. לא, אלה פריטי LR(0) וקיים מצב כזה בפיתוח LR(0) עבור הדקדוק G_2
ה. לא, אלה בכלל לא פריטי LR(0) או LR(1)

שאלה 14 (4 נק')

יהי הדקדוק הבא:

$$S \to E$$
$$E \to \underline{num} \mid E\ \underline{@}\ E \mid \underline{(}\ E\ \underline{)}$$

לאיזו מילה בשפה של הדקדוק קיים יותר מעץ גזירה אחד?

א. num @ num @ num
ב. num @ (num)
ג. (num @ num) @ (num @ num)
ד. ε
ה. אין מילה כזו בשפה של הדקדוק.

חלק ב' - שאלות פתוחות (50 נק')

שאלה 1: ייצור קוד (20 נק')

הוסיפו לשפת FanC מבנה בקרה חדש, `from_until`:

```
int i = 1;
from_until(3 ; 5 ; 3 < 4, 7 < 8, 8 < 9, 2 < 4, 5 < 8, 4 > 3) {
    i = i + 2;
    print("%d", i);
}
//prints 3 5 7
```

בתוך הסוגריים של `from_until` יש שני ביטויי E ואחריהם רשימה לא ריקה של ביטויים בוליאניים. הקוד בודק את ערכי הביטויים E ולאחריהם עובר על רשימת התנאים החל מהתנאי הבוליאני במקום של ערך הביטוי הראשון ועד התנאי הבוליאני במקום של ערך הביטוי השני כולל ולאחר מכן יוצאים מהלולאה.

אם תנאי שנבדק מתקיים נכנסים לגוף הלולאה ובסופה ממשיכים לבדוק את הביטויים הבוליאניים החל מהביטוי בו עצרנו. אם תנאי שנבדק לא מתקיים יוצאים מהלולאה.

חשוב לשים לב שב-`from_until`:

- ניתן להניח כי ערך הביטוי הראשון קטן מערך הביטוי השני.
- ניתן להניח כי ערכי הביטויים יהיו לכל היותר כמספר התנאים הבוליאניים.
- מספור התנאים הבוליאניים מתחיל מ-1.

הדקדוק שאיתו ממומש `from_until` הוא:

$$S \rightarrow \underline{from\_until}\ \underline{(}\ E_1\ \underline{;}\ E_2\ \underline{;}\ B\_LIST\ \underline{)}\ \underline{\{}\ S_1\ \underline{\}}$$

$$B\_LIST \rightarrow B\_LIST_1\ \underline{,}\ B$$

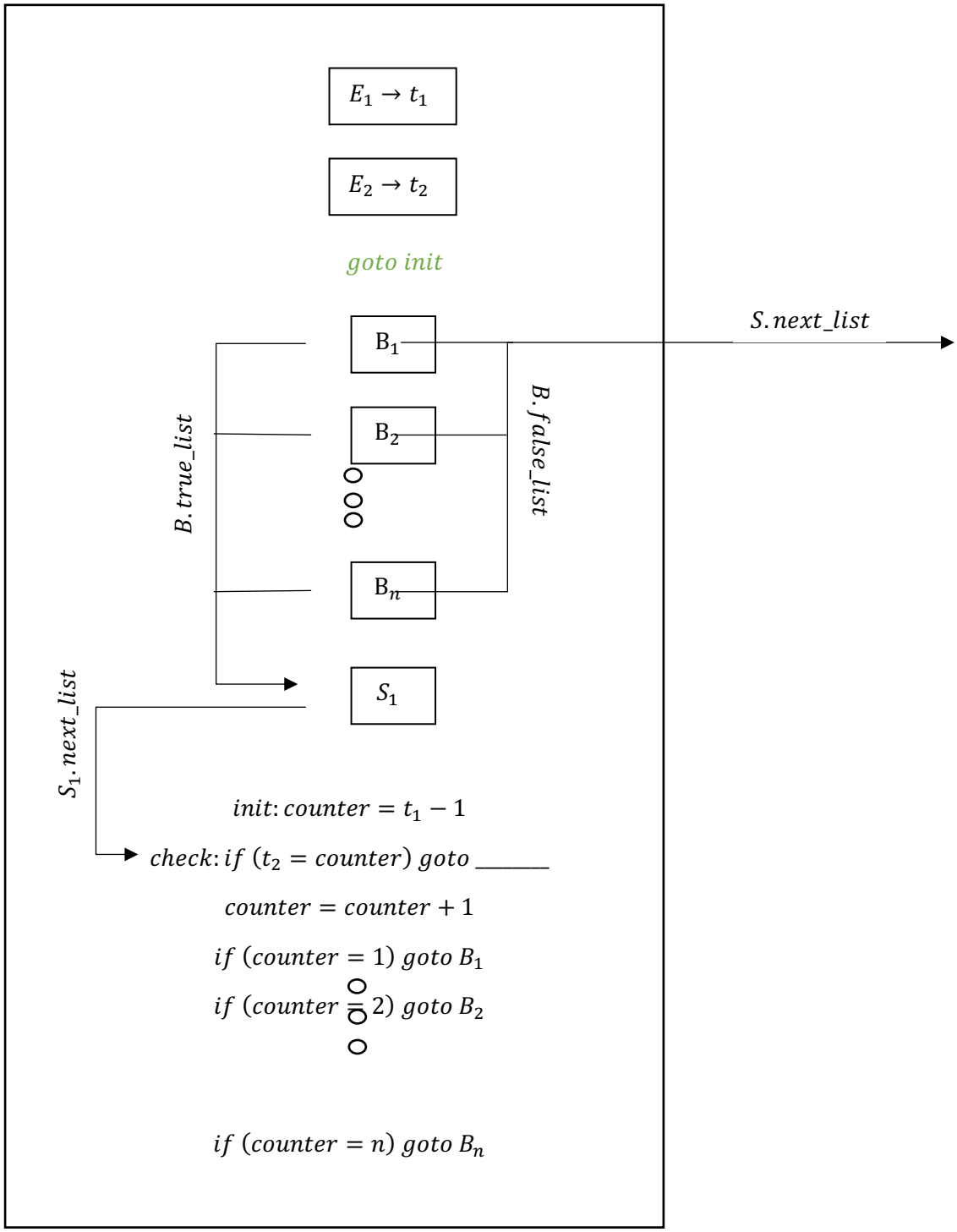
$$B\_LIST \rightarrow B$$

שימו לב:

- ניתן להניח שבקוד לא יהיו שגיאות קומפילציה
- אין לשנות את הדקדוק פרט להוספת מרקרים M,N
- ניתן להשתמש במרקרים M, N שנלמדו בכיתה בלבד
- למשתנים S, E, B ישנן התכונות שהוגדרו בכיתה בלבד
- למשתנים S, E, B ישנם כללי גזירה פרט לאלה המוצגים בשאלה
- אין חשיבות לסקופים
- אסור להשתמש במשתנים גלובליים
- המשתנה S תמיד יכיל $nextlist$

1. 8)נק') הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות הסמנטיות שאתם משתמשים בהן עבור כל משתנה .
2. 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 .

פתרון:



<u>תכונות סמנטיות:</u> עבור E, B, S נשתמש בתכונות הסמנטיות הרגילות שנלמדו בכיתה

עבור B_list נשמור את התכונות הסמנטיות true_list, false_list ו quad_list כאשר true_list ו false_list הן רשימות של כתובות של מקומות בקוד שצריך למלא בהם את הכתובת אליה יש לקפוץ במקרה שהערך של B_list הוא true או false בהתאמה ו quad_list היא רשימה של הכתובות שבהן מתחיל הקוד של כל B_i כאשר $1 \leq i \leq n$.

(פירוט התכונות שנלמדו בכיתה עבור כל משתנה:

E – place (המשתנה הזמני אשר הוקצה לביטוי)

B – true_list ו false_list (רשימות של כתובות של מקומות בקוד שצריך למלא בהם את הכתובת אליה יש לקפוץ במקרה שהערך של B הוא true או false בהתאמה)

S – next_list (רשימה של כתובות של מקומות בקוד שצריך למלא בהם את הכתובת אליה יש לקפוץ לאחר סיום הקוד של S))

<u>טעויות נפוצות-</u>

1. היו סטודנטים שרשמו ש E_1 ו E_2 הם ערכים שידועים בזמן קומפילציה.
הטענה הנכונה היא שהערך שנשמר ב E_1 וב E_2 לא ידוע בזמן קומפילציה, כן ידוע באיזה משתנה הם שמורים ומשתנה זה שמור בתכונה הסמנטית $E.place$.
2. היו סטודנטים שיצרו פריסה אשר תלויה בערך של t_1 ו t_2, כלומר, יצרו משהו כזה

$$\boxed{\begin{array}{c} if\ (t1 = 1) goto\ B_1 \\ if\ (t1 = 2)\ goto\ B_2 \\ \\ if\ (t1 = t2)\ goto\ B_{t_2} \end{array}}$$

זה לא נכון כיוון שהערך של t_2 לא ידוע בזמן קומפילציה ולכן לא ניתן לדעת בזמן ייצור הקוד עד לאיפה להדפיס.

3. היו סטודנטים שכתבו קוד שלא נוצר על ידי המרקרים N או M בין המשתנים. נשים לב כי דבר זה אינו נכון כיוון שפריסת הקוד מייצגת את הקוד שנוצר בbuffer עבור S. כיוון שהקוד שאנו רושמים בS נוצר לאחר שהקוד של E, B ו B_list כבר נכתב בbuffer לא ניתן לרשום קוד בין המשתנים ללא שימוש במרקרים נוספים (אשר לפי הוראות התרגיל לא היה ניתן להוסיף).

<u>סעיף ב-</u>

```
S -> from_until (E1 ; E2 N ; B_list) {M S1}
{
    bp(N.next_list, next_quad());
    bp(B_list.true_list, M.quad);
    counter = new_temp();
    emit(counter || " = " || E1.place || " - 1 ");
    S.next_list = merge(B_list.false_list, make_list(next_quad()));
    bp(S1.next_list, next_quad());
    emit("if (" || E2.place || " = " || counter || ") goto ____");
    emit(counter || " = 1 + " || counter);
    int i = 1;
    while (!(B_list.quad_list.empty()))
    {
        quad = B_list.quad_list.pop();
        emit("if (" || counter || " = " || i || ") goto " || quad);
        i++;
    }
}
```

שאלה 2: אנליזה סטטית (30 נק')

במפקדת החלל האמריקנית הוחלט לפתח קומפיילר לתוכניות חלל, ועל קפטן פייק הוטל לתכנן אופטימיזציה עבור לולאות. פייק הבחין, שרוב הלולאות בתוכניות שהגיעו לידיו מתבצעות מספר חסום של פעמים, וכי החסם ניתן לגילוי בזמן הקומפילציה.

נכנה בשם "לולאת פייק" קטע קוד ביניים (IR) אשר מקיים את כל שלושת התנאים הבאים:

1. הקוד הוא לולאה. (כזכור, ההגדרה של לולאה בקוד ביניים היא קבוצה של בלוקים בסיסיים כך שכל הקפיצות מגוף הלולאה אל הבלוק הראשון — כותרת הלולאה — נשלטות על ידיו.)
2. בלולאה יש משתנה אשר חסום מלמעלה על ידי הפרמטר B=16. (הניחו שאפשר לקנפג את B על ידי דגל לקומפיילר.)
3. המשתנה הזה גדל בהכרח בכל איטרציה של הלולאה.

כל המשתנים בייצוג הביניים של פייק הם מסוג unsigned int, והאופרטורים הם +, –, * (חיבור, חיסור וכפל). בנוסף, יש בשפה תוויות, קפיצות מותנות וקפיצות בלתי מותנות, כרגיל.

א. (15 נק') הציעו לקפטן שיטה, המשלבת **שתי** אנליזות סטטיות, לצורך זיהוי לולאות פייק. האנליזה הראשונה תבדוק את תנאי מספר 2, והאנליזה השנייה תבדוק את תנאי מספר 3.

לכל אנליזה, הגדירו את הדומיין (כסריג), פעולת ה join ופונקציות המעבר המתאימות.

הוכיחו שפונקציות המעבר מונוטוניות.

הסבירו כיצד הקומפיילר ישתמש בתוצאות שתי האנליזות כדי לסווג את הלולאות.

<u>הנחות והנחיות</u>:

- הניחו כי פייק כבר הריץ את חישוב הבלוקים השולטים ויודע היכן נמצאות הלולאות, היכן נמצאת הכותרת של כל לולאה והיכן מסתיים גוף הלולאה. בנוסף, לצורך תמיכה באנליזות סטטיות, הוא שינה את קוד הביניים כך שבלוק המהווה כותרת של לולאה מתחיל תמיד בפקודה מיוחדת בשם "loop-header", אשר מעבר לכך אין לה השפעה על ביצוע התוכנית.
- בתוכניות יכולות להופיע קריאות מהזיכרון וקריאות לפונקציות. יש להניח שבמקרים אלה הערך שנקרא/מוחזר הוא ערך שלם כלשהו אשר איננו ידוע בזמן הקומפילציה.
- לאנליזה השנייה מותר להשתמש בתוצאות של האנליזה הראשונה.
- ניתן להשתמש בכל אנליזה שנלמדה בקורס בציון שמה בלבד ואין צורך לכתוב את ההגדרות במחברת.
- <u>שימו לב</u>: בשאלה זו **אינכם** נדרשים לבצע אופטימיזציה כלשהי, אלא לממש את *האנליזה המתוארת בלבד*.

כמה דוגמאות להמחשה:

loop1: loop-header **if** i < 20 **goto** exit1 i := i + 2 print(i) **goto** loop1 exit1:	loop2: loop-header **if** i < 10 **goto** exit2 t := read_int() **if** t == 0 **goto** else2 i := i + 1 **goto** loop2 else2: i := i + 2 **goto** loop2 exit2:	loop3: loop-header **if** i < 10 **goto** exit3 t := read_int() **if** t == 0 **goto** else3 i := i + 1 **goto** loop3 else3: i := i – 1 **goto** loop2 exit3:	loop4: loop-header **if** i < 10 **goto** exit4 i := inc_func(i) t1 := a + i t2 := *t1 print(t2) **goto** loop4 exit4:
לא פייק	כן פייק	לא פייק	לא פייק

ב. (10 נק') מפקדו הישיר של קפטן פייק הטיל ספק בכך שהאנליזה שמימש נאותה, זאת מכיוון שלא כתבתם עבורה את פונקציות האבסטרקציה (α) והקונקרטיזציה (γ).

פייק הצייתן מיהר והשלים את הפונקציות עבור האנליזה הראשונה (זו שבודקת את החסמים) אך התקשה להגדיר אותן עבור האנליזה השנייה (זו שבודקת שמשתני הלולאה גדלים בכל איטרציה). כדי לגשת לבעיה, הוא החליט לשנות את קוד הביניים כך שבכותרת הלולאה, מיד לאחר הפקודה המיוחדת loop-header, תופענה פקודות המעתיקות את כל המשתנים המקומיים למשתני עזר: כל משתנה v יועתק למשתנה-עזר מתאים $v0.

למשל, במקום:	ייכתב עכשיו:

`myloop:`
`  loop-header`
`  if i >= n goto exitloop`
`  i = i + 1`
`  goto myloop`
`exitloop:`
`  print("bye")`
	`myloop:`
	`  loop-header`
	`  **$i0 = i**`
	`  **$n0 = n**`
	`  if i >= n goto exitloop`
	`  i = i + 1`
	`  goto myloop`
	`exitloop:`
	`  print("bye")`

למרבה הצער, עד שהספיק לסיים את השינוי הזה בקומפיילר הגיע זמן ארוחת הערב והוא נאלץ לצאת מבלי שסיים את הגדרת הפונקציות. הגדירו עבורו את הפונקציות הללו בהסתמך על התוכניות המתוקנות.

שימו לב שהמצבים הקונקרטיים של כל תוכנית כוללים עכשיו השמות למשתני-העזר; ניתן וצריך להשתמש בערכים אלה בהגדרות שלכם.

ג. (5 נק') נסחו מִזְכָּר (memo) קצר למפקד, המנמק מדוע **בכל ריצה** של לולאה שזוהתה כלולאת פייק על ידי האנליזות יתבצע גוף הלולאה לכל היותר B פעמים לפני שיוצאים מהלולאה. בניסוח הסתמכו על הפונקציות מהסעיף הקודם.

פתרון:

א.

ניתוח טווחים מספריים של משתנים בתכנית ניתן לעשות בעזרת interval analysis, אנליזה שנלמדה בשיעור abstract interpretation. בזכור, האנליזה מפיקה עבור כל משתנה בתכנית ועבור כל מיקום בתכנית מקטע [a, b].

כדי לבדוק את תנאי 2 (קיום של משתנה החסום מלמעלה על ידי B) יש להסתכל על **הבלוק הראשון בגוף הלולאה**, ולזהות משתנה או משתנים שעבורם b < B.

הערה: פתרון שמסתכל על תוצאת האנליזה בכותרת הלולאה (loop header) איננו מדויק, מכיוון שבמקרים שבהם תנאי הלולאה הוא false אין נכנסים ללולאה. זהו אי-דיוק קטן ולא הורדו על כך נקודות בבדיקה.

אנליזה המתאימה לבדיקת תנאי 3 (קיום של משתנה אשר בכל איטרציה ערכו גדל בהכרח) היא אנליזה המבוססת על interval analysis כאשר מחילים אותה על ההפרשים בין ערך המשתנה לבין הערך שהכיל בתחילת כותרת הלולאה. הדומיין הוא אותו סריג של intervals. פונקציות המעבר הן:

$[\![\text{loop-header}]\!]^{\#}\sigma^{\#} = \{v_1 \mapsto 0, v_2 \mapsto 0, \ldots \text{ (for all } v_i \in \text{Vars) } \ldots\}$

$[\![v := v + e]\!]^{\#}\sigma^{\#} = [\![v := e + v]\!]^{\#}\sigma^{\#} = \sigma^{\#}[v \mapsto \sigma^{\#}(v) + A_1[e]]$

$[\![v := v - e]\!]^{\#}\sigma^{\#} = \sigma^{\#}[v \mapsto \sigma^{\#}(v) - A_1[e]]$

$[\![v := v * e]\!]^{\#}\sigma^{\#} = [\![v := e * v]\!]^{\#}\sigma^{\#} = \sigma^{\#}[v \text{ -> } \sigma^{\#}(v) + (A_1[e] - 1) * A_1[v]]$

$[\![v := \_]\!]^{\#}\sigma^{\#} = \sigma^{\#}[v \mapsto T]$ (all other assignments)

$[\![s]\!]^{\#}\sigma^{\#} = \sigma^{\#}$ (all other statements)

הערות:

- יש לזכור שמדובר בייצוג ביניים, לכן e בהשמות לעיל יכול להיות רק משתנה או קבוע. מכיוון שמדובר במעקב אחר הפרשים, יש שימוש רק להשמות שבהן המשתנה שמופיע בצד שמאל נמצא גם בצד ימין כאחד האופרנדים. לכן אין תועלת בהגדרה של #[e] סמנטיקה אבסטרקטית של ביטויים.)
- במקום איפוס המשתנים ב loop-header, אפשר להריץ את האנליזה רק על גוף הלולאה כאשר המצב המאופס הוא המצב ההתחלתי. תשובה שמתעלמת מההיבט הזה אינה שלמה אבל לא הורדו נקודות על כך.
- הסמנטיקה של הכפל היא מורכבת ומבוססת על השקילות (v * e = v + v * (e – 1. התקבלו גם תשובות פחות מדויקות כולל T.
- בפתרון השתמשנו בסימון A1[e] עבור הערך האבסטרקטי (בדומיין האינטרוולים) של הביטוי e **באותה נקודה בתכנית** שבא נמצאת ההשמה המדוברת.
- פתרון פשוט יותר יכול להסתפק במעקב אחר הסימן של ההפרש במקום אחר הטווח המלא. אבל חייבים להשתמש בדומיין הסימנים שבו +, 0, – אינם בני השוואה (נמצאים "באותה רמה"). בדומיין שבו + ⊑ 0 לא ניתן יהיה להפריד בין משתנה שגדל בערכו לבין משתנה שערכו נשמר.
- שמו של הקפטן בשאלה הוא "Pike" לא."fake"

איך הקומפיילר ישתמש בתוצאה: צריך להסתכל על המצב **בסוף גוף הלולאה**, כלומר ב out(b) של הבלוק b האחרון בגוף הלולאה (מה שקראנו לו בשיעור, "זנב" הלולאה). שם צריך לבדוק האם הטווח שחושב [a, b] הוא כזה ש.a > 0

ב.

ראשית יש להבין את הצורך במשתנה $ v0 בהקשר של הסעיף הזה. מצב אבסטרקטי מהצורה {v ↦ [a, b]} לא מספק מידע על הערכים הקונקרטיים האפשריים של המשתנה v, מכיוון שערכים אלה תלויים במה שהיה ערכו של v בתחילת האיטרציה של הלולאה. התוספת של פייק מאפשרת לכלול ערכים מקוריים אלה במצב הקונקרטי הנוכחי. כלומר, המשמעות של ההשמה האבסטרקטית {v ↦ [a, b]} היא שההפרש v – $v0 נמצא בטווח בין a ל.b

ניתן להגדיר פונקצית עזר beta מעל **הזוג**.v, $v0

$\beta(v, \$v0) = [v - \$v0, v - \$v0]$

$\beta(s\#) = \{v_1 \mapsto \beta(\sigma^{\#}(v_1), \sigma^{\#}(\$v_1 0)), v_2 \mapsto \beta(\sigma^{\#}(v_2), \sigma^{\#}(\$v_2 0)), \dots \}$

ואז משתמשים בהגדרה הסטנדרטית של α ו γ בעזרת.β

ג.

נסתכל על ריצה של לולאת פייק, כאשר המשתנה שעומד בתנאים יסומן ב .c נסתכל על סדרת מצבים קונקרטיים בריצה כלשהי של הלולאה, $\sigma_1, \dots, \sigma_n$, בכל ביקור ב header של הלולאה.

הסדרה של הזוגות:

$\langle \sigma_1 c, \sigma_1 \$c0 \rangle, \langle \sigma_2 c, \sigma_2 \$c0 \rangle, \dots, \langle \sigma_n c, \sigma_n \$c0 \rangle$

מתארת את ערכי c, $c0 בכל אחד מהמצבים. מכיוון שכל איטרציה מתחילה בהשמה c0 := c$, אנחנו יודעים כי

$\sigma_2 \$c0 = \sigma_1 c \; ; \; \sigma_3 \$c0 = \sigma_2 c \; ; \dots$

וכך בכל מצב לעומת קודמו. על-פי פונקצית γ שהגדרנו בסעיף ב, בכל אחד מהמצבים מתקיים $\sigma_i c - \sigma_i \$c0 \in [a, b]$, כאשר a > 0. בפרט, $\sigma_i c > \sigma_i \$c0$, לכן $\sigma_i c - \sigma_{i-1} c$. באינדוקציה, $\sigma_n c \geq n$. מכיוון שבגוף הלולאה הוכחנו ש $c \leq B$, הרי שאם n > B בהכרח התוכנית תצא מהלולאה (אם היא נכנסת ללולאה נקבל c > B בסתירה לנאותות האנליזה הראשונה.)

הערה: הכוונה בשאלה הייתה להסתמך על הפונקציות α, γ בנימוק הפורמלי אך גם תשובות פחות פורמליות התקבלו.

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

הגדרה: דקדוק G הוא LL(1) אם ורק אם 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:

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

Abstract Interpretation

בהינתן סריג L עם יחס סדר חלקי $\sqsubseteq$ ופעולת join $\sqcup$:

מצב אבסטרקטי הוא $\sigma^{\#} \in A = (Var \to L)$

ערך אבסטרקטי של משתנה $x \in \mathrm{Var}$ בהינתן מצב אבסטרקטי: $\sigma^{\#}(x)$

סמנטיקה אבסטרקטית של ביטוי e: $[\![e]\!]^{\#} : A \to L$

סמנטיקה אבסטרקטית של משפט s: $[\![s]\!]^{\#} : A \to A$

ערך אבס' של ביטוי e או מצב אבס' אחרי משפט s בהינתן מצב אבס' קודם: $[\![e]\!]^{\#}\sigma^{\#}$ / $[\![s]\!]^{\#}\sigma^{\#}$

מצב אבס' אחרי השמה של ערך אבס' $v \in L$ למשתנה $x \in \mathrm{Var}$: $\sigma^{\#}[x \mapsto v]$

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
AUTO	auto
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	}
ASSIGN	=
RELOP	==|<=|>=|<|>|!=
BINOP	+|-|*|/
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. $Statements \rightarrow Statement$
13. $Statements \rightarrow Statements\ Statement$
14. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
15. $Statement \rightarrow Type\ ID\ SC$
16. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
17. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow Call\ SC$
19. $Statement \rightarrow RETURN\ SC$
20. $Statement \rightarrow RETURN\ Exp\ SC$
21. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
22. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
23. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
24. $Statement \rightarrow BREAK\ SC$
25. $Statement \rightarrow CONTINUE\ SC$
26. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
27. $Call \rightarrow ID\ LPAREN\ RPAREN$
28. $ExpList \rightarrow Exp$
29. $ExpList \rightarrow Exp\ COMMA\ ExpList$
30. $Type \rightarrow INT$
31. $Type \rightarrow BYTE$
32. $Type \rightarrow BOOL$
33. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
34. $Exp \rightarrow Exp\ BINOP\ Exp$
35. $Exp \rightarrow ID$
36. $Exp \rightarrow Call$
37. $Exp \rightarrow NUM$
38. $Exp \rightarrow NUM\ B$
39. $Exp \rightarrow STRING$
40. $Exp \rightarrow TRUE$
41. $Exp \rightarrow FALSE$
42. $Exp \rightarrow NOT\ Exp$
43. $Exp \rightarrow Exp\ AND\ Exp$
44. $Exp \rightarrow Exp\ OR\ Exp$
45. $Exp \rightarrow Exp\ RELOP\ Exp$
46. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$