2022-10-21

מבחן סוף סמסטר – מועד ב'

פתרון

מרצה אחראית: ד"ר הילה פלג

מתרגלים: מתן פלד, תומר כהן, אלון קיטין, מתן ממיסטולוב

הוראות:

א. בטופס המבחן 14 עמודים, מתוכם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
ב. משך המבחן שלוש שעות (180 דקות).
ג. כל חומר עזר חיצוני אסור לשימוש.
ד. בשאלות הפתוחות, ניתן לציין לגבי סעיף או שאלה "לא יודע/ת". תשובה זו תזכה ב-20% מהניקוד. תשובות שגויות לא יזכו בניקוד.
ה. בשאלות הסגורות, אם לדעתכם יש יותר מתשובה נכונה אחת, בחרו את התשובה הנכונה ביותר.
ו. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.
ז. אין צורך להגיש את טופס מבחן זה בתום הבחינה.
ח. את התשובות לשאלות הסגורות יש לסמן בטופס התשובות הנפרד בלבד. את התשובות לשאלות הפתוחות יש לכתוב במחברת הבחינה.
ט. ודאו כי אתם מגישים טופס תשובות ומחברת בחינה בלבד.

בהצלחה!

חלק א' - שאלות סגורות (50 נק')

שלבי קומפילציה

נתון קטע הקוד הבא בשפת FanC:

```
1: int somefunc(int s) {
2:     int x0 = s / 2;
3:     if (x0 != 0) {
4:         int x1 = (x0 + s / x0) / 2;
5:         while (x1 < x0) {
6:             x0 = x1;
7:             x1 = (x0 + s / x0) / 2;
8:         }
9:         return x0;x1
10:    }
11:    return s;
12:}
```

בסעיפים הבאים (שאלות 1-3) מוצגים שינויים (**בלתי תלויים**) לקוד של הקטע הרשום מעלה.
עבור כל שינוי ציינו את השלב המוקדם ביותר שבו נגלה את השגיאה (2 נק' לשאלה).

הנחיות:
1. ניתן להניח שאין שגיאות בקטע הקוד לפני השינוי
2. ניתן להתעלם משגיאות פוטנציאליות בשאר התוכנית

שאלה 1 (2 נק')

נחליף את הביטוי 's' בשורה 11 בביטוי 'x1'. semantic

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

שאלה 2 (2 נק')

נמחק את כל סימני הנקודה-פסיק (';') הקיימים בתוכנית. syntax תחבירי

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

שאלה 3 (2 נק')

נחליף את הביטוי 'x0' בשורה 9 בביטוי 'x1'. No error

א. אין שגיאה
ב. שגיאה בניתוח תחבירי
ג. שגיאה בניתוח לקסיקלי

ד. שגיאה בניתוח סמנטי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה 4 (2 נק')

מקמפלים את התוכנית עם קומפיילר שיש בו באג בו טיפוס התוצאה של פעולות אריתמטיות (+, -, *, /) הוא byte במקום int. אם התוצאה גדולה מדי בשביל byte אז יבוצע truncation – הפעולה תזרוק את הביטים הגבוהים. באיזה שלב קומפילציה תתגלה שגיאה אם בכלל?
הערה: לצורך שאלה זו, תוצאה מספרית שונה לעומת קומפיילר ללא באג לא נחשבת כשגיאה. No err

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

שאלה 5 (2 נק')

באיזה שלב מתבצעת ההחלטה האם לבצע קודם את החילוק ב-x0 או את החיבור בשורה 4? Left/Right הרמה כי

א. ניתוח תחבירי
ב. ניתוח לקסיקלי
ג. ניתוח סמנטי
ד. ייצור קוד
ה. זמן ריצה

שאלה 6 (5 נק')

כדי להתמודד עם חולשות אבטחה כגון Spectre , התבקשנו להוסיף לקומפיילר תמיכה במבנה בקרה חדש בשם **barrier** האומר למעבד שאסור לו לבצע speculative execution בקוד המוכל במבנה. המימוש הוא לעשות emit לפקודת שפת מכונה ספציפית שאומרת את הנ"ל למעבד, וגם לדאוג להתנהגות דומה בכל אופטימיזציה שהקומפיילר מבצע.
דוגמאת שימוש:

```
barrier {
    if (address <= MAXIMUM_ALLOWED) {
        value = read_memory(address);
    } else {
        value = -1;
    }
}
write_memory(OUTPUT, value);
```

ידוע לנו כי אין צורך בשינויים בשלב זמן הריצה, מכיוון שהמעבד כבר מממש את פקודת שפת המכונה המדוברת וכמו כן אנחנו לא משתמשים ב-JIT.
באיזה שלב **לא** נצטרך לערוך שינויים?

א. ניתוח סמנטי
ב. ניתוח תחבירי
ג. ניתוח לקסיקלי
ד. ייצור קוד
ה. נצטרך לשנות את כל שלבי הקומפילציה (מלבד זמן ריצה כאמור)
ו. לא נצטרך לשנות אף שלב קומפילציה

שאלה 7 (5 נק')

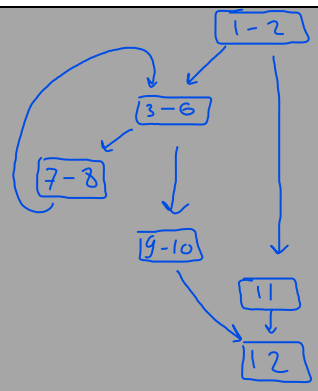
נרצה להוסיף תמיכה בדיבאגרים לקומפיילר שלנו. תמיכה כזו תתבטא בזה שלא נבצע אופטימזציות כאשר אנחנו מקמפלים במצב דיבאג.
מהו שלב הקומפילציה המוקדם ביותר אותו נצטרך לשנות? code gen

א. ייצור קוד
ב. ניתוח תחבירי
ג. ניתוח סמנטי
ד. ניתוח לקסיקלי
ה. זמן ריצה

אופטימיזציות

נתון הקוד הבא בשפת ביניים, המייצג את הגירסה המקומפלת (ע"י קומפיילר תקין) של somefunc מהשאלה הקודמת. הניחו כי s הוא פרמטר, nop היא פקודה שלא מבצעת דבר, וכי הערך שנמצא במשתנה ששמו כשם הפונקציה הוא שמוחזר בסוף הפונקציה (כלומר, כך מבצעים return).

```
1:  x0 = s / 2
2:  if x0 == 0 goto 11
3:  t0 = s / x0
4:  t1 = x0 + t0
5:  x1 = t1 / 2
6:  if x1 >= x0 goto 9
7:  x0 = x1
8:  goto 3
9:  somefunc = x0
10: goto 12
11: somefunc = s
12: nop
```



6 Blocks
7 edges

שאלה 8 (5 נק')

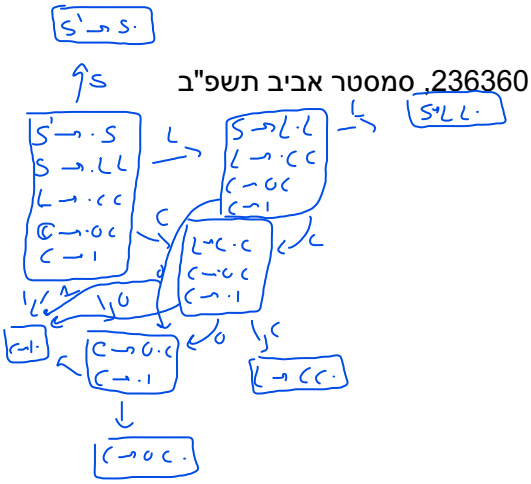
ציירו את ה-CFG של קוד הביניים הנתון. מה מספר הצמתים והקשתות שבגרף?

א. 6 צמתים ו-7 קשתות
ב. 5 צמתים ו-6 קשתות
ג. 6 צמתים ו-6 קשתות
ד. 5 צמתים ו-7 קשתות

שאלה 9 (5 נק')

אם נסתכל על הקוד נוכל לראות שאין צורך גם ב-x0 וגם ב-somefunc. את somefunc אנחנו צריכים כדי לבצע return, אבל אם נחליף את כל המופעים של x0 ב-somefunc, אז שורה 9 תהפוך להיות השמה של somefunc לעצמה, ואז נוכל להסיר אותה. איזו אופטימזציה שלמדנו בכיתה תהפוך את שורה 1 להיות somefunc = s / 2?

א. אף תשובה אינה נכונה
ב. copy propagation
ג. constant propagation
ד. common subexpression elimination
ה. constant folding



דקדוקים

עאבד וטרוי דנים בדקדוק G_1 בתוכנית הבוקר שלהם:

$$S \to L\ L$$
$$L \to C\ C$$
$$C \to 0\ C \mid 1$$

שאלה 10 (3 נק')

טרוי רוצה לדעת לאיזה אוטומט פרפיקסי של הדקדוק G_1 יש את מספר המצבים הקטן ביותר. בחרו את התשובה המדוייקת ביותר:

א. LR(0) ו-SLR
ב. LR(0)
ג. SLR
ד. LR(1)
ה. LR(0), SLR, ו-LR(1)

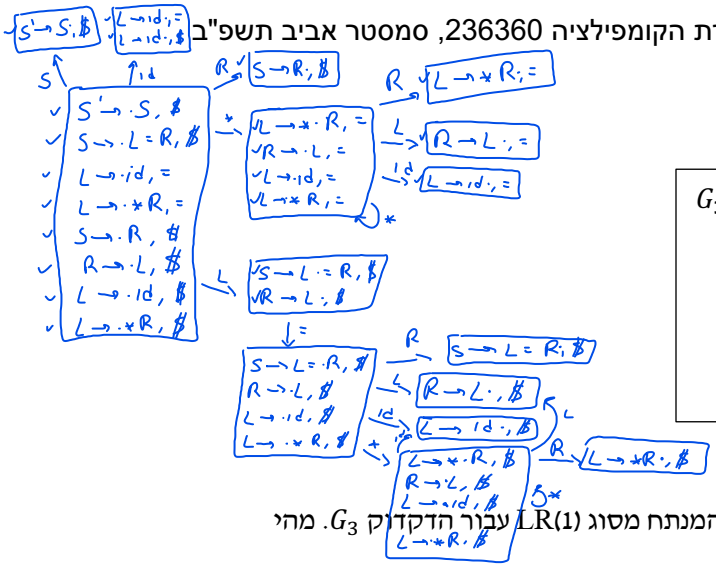
שאלה 11 (3 נק')

אנני הגיעה בתור אורחת לתוכנית של טרוי ועאבד. היא הוסיפה ל-G_1 בדיקה סמנטית של כמות ה-"0"-ים וה-"1"-ים, המוודאת שבין כל תו של "1", כולל לפני ה-"1" הראשון, קיימת כמות שווה של "0"-ים (לדוגמא, חוקי: 01010101, לא חוקי: 00110101). היא מימשה זאת באופן הבא:

$S \to L_1\ L_2$	$\{if\ L_1.x \neq L_2.x\ then\ reject\}$
$L \to C_1\ C_2$	$\{if\ C_1.x = C_2.x\ \ then\ L.x = C_1.x\ else\ reject\}$
$C \to 0\ C_1$	$\{C.x = C_1.x + 1\}$
$C \to 1$	$\{C.x = 0\}$

מהי הטענה הנכונה?

א. אנני השתמשה רק בתכונות נוצרות
ב. אנני השתמשה גם בתכונות נוצרות וגם בתכונות נורשות, **ואין** תלות מעגלית בין התכונות
ג. אנני השתמשה גם בתכונות נוצרות וגם בתכונות נורשות, **ויש** תלות מעגלית בין התכונות
ד. אנני השתמשה רק בתכונות נורשות
ה. אף תשובה לא נכונה

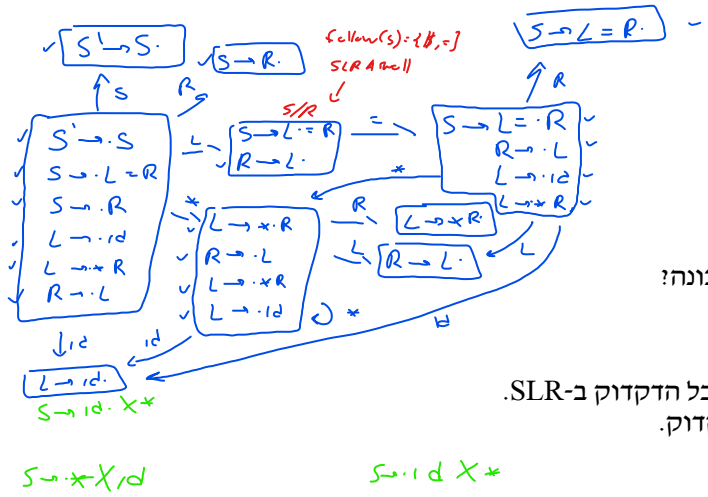


ג'ף רצה להעסיק את חבריו ולכן נתן להם את הדקדוק G_3:

G_3:
$S \to L = R$
$S \to R$
$L \to id$
$L \to * R$
$R \to L$

שאלה 12 (3 נק')

ג'ף ביקש מהקבוצה לדעת כמה מצבים ישנם באוטומט פריפיקסי של המנתח מסוג LR(1) עבור הדקדוק G_3. מהי התשובה הנכונה?



א. 14
ב. 13
ג. 12
ד. 15
ה. הדקדוק אינו ב-LR(1)

שאלה 13 (3 נק')

בריטה טוענת שהדקדוק של ג'ף (G_3) הוא גם ב-LR(0), מהי הטענה הנכונה?

א. בריטה טועה, קיים קונפליקט shift/reduce אחד בדקדוק.
ב. בריטה טועה, קיים קונפליקט reduce/reduce אחד בדקדוק.
ג. בריטה טועה, קיים קונפליקט shift/reduce אחד בדקדוק, אבל הדקדוק ב-SLR.
ד. בריטה טועה, קיימים שני קונפליקטים של shift/reduce בדקדוק.

שאלה 14 (4 נק')

הדיקן מציע רישום מוקדם למי שיענה נכונה על השאלה:

יהי דקדוק $G = (V, T, P, S)$, נגדיר את הדקדוק $G' = (V, T, P \cup \{S \to aXb\}, S)$ כאשר $a, b \in T, X \in V$

מהי הטענה הנכונה ביותר?

א. אם G ב-LR(1) וגם לכל $Y \in V$, $a \notin first(Y)$ אזי G' גם ב-LR(1)
ב. אם G ב-SLR וגם לכל $Y \in V$, $a \notin first(Y)$ אזי G' גם ב-SLR
ג. אם G ב-LR(0) וגם **קיים** $Y \in V$, $a \notin first(Y)$ אזי G' גם ב-LR(0)
ד. אם G ב-SLR וגם לכל $Y \in V$, $a \notin first(Y), b \notin follow(Y)$ אזי G' גם ב-SLR
ה. יותר מתשובה אחת נכונה

שאלה 15 (4 נק')

שירלי נתקלת בבעיה הבאה בזמן הפתיחה של חנות הסנדוויצ'ים שלה: יהי G דקדוק של שפה **סופית** L, מהי הטענה הנכונה ביותר?

א. G לא בהכרח ב-LL(1), אבל בהכרח קיים דקדוק שמקבל את L שניתן לבנות עבורו מנתח LL(1)
ב. G לא בהכרח ב-LL(1), ואם G לא ב-LL(1) אז לא ניתן לבנות מנתח LL(1) לאף דקדוק שמקבל את L
ג. G לא בהכרח ב-LL(1), אבל יתכן שקיים דקדוק שמקבל את L שניתן לבנות עבורו מנתח LL(1) גם אם G לא ב-LL(1)
ד. G בהכרח ב-LL(1) ולכן ניתן לבנות לו מנתח LL(1)

חלק ב' - שאלות פתוחות (50 נק')

שאלה 1 : ייצור קוד (20 נק')

התבקשתם להרחיב את שפת FanC עם מבנה הבקרה חדש, fancy_while :

```
int i = 3;
fancy_while (5; 4 < 5, 3 > 5, i > 1) {
    i--;
    print("%d", i);
} //prints 2 1
```

בתוך הסוגריים של fancy_while יש ביטוי E ואחריו רשימה לא ריקה של ביטויים בוליאניים. הקוד בודק את ערך הביטוי E ולאחריו עובר על רשימת התנאים לפי הייצוג הבוליאני שלו, כאשר התנאי ה-i ייבדק רק אם הביט ה- i דולק. אם תנאי שנבדק מתקיים, הבדיקה תעבור אל התנאי של הביט הדולק הבא. אם תנאי שנבדק לא מתקיים, הלולאה תצא. בדוגמא למעלה במספר 5 רק הביט הראשון והשלישי דולקים ולכן רק התנאי הראשון והשלישי יבדקו.

אם כל התנאים שנבדקים מתקיימים נכנסים לגוף הלולאה ובסופה דוגמים מחדש את הביטוי E ולפיו בודקים את רשימת התנאים באותה הצורה.

חשוב לשים לב שב-fancy_while :

- הערך של ביטוי E עלול להשתנות בתוך גוף הלולאה בלבד ולא בתנאים.
- אם ערך הביטוי 0 אזי באופן ריק מתקיימים כל התנאים ויש להיכנס לגוף הלולאה.
- אם בייצוג הטיפוס של E אין מספיק ביטים עבור מספר התנאים ברשימת התנאים, ניתן להניח כי ערך הביט ה- i עבור כל i שחורג מגודל הטיפוס הוא 0.
- ערך הביטוי יכול להכיל ביטים דולקים באינדקסים שלא קיימים תנאים עבורם (לדוגמא עבור הערך 120b עם רשימה של 2 תנאים). חשוב לשים לב שבמימוש סביר של המבנה לא תהיה לכך משמעות.

הדקדוק שאיתו ממומש fancy_while הוא :

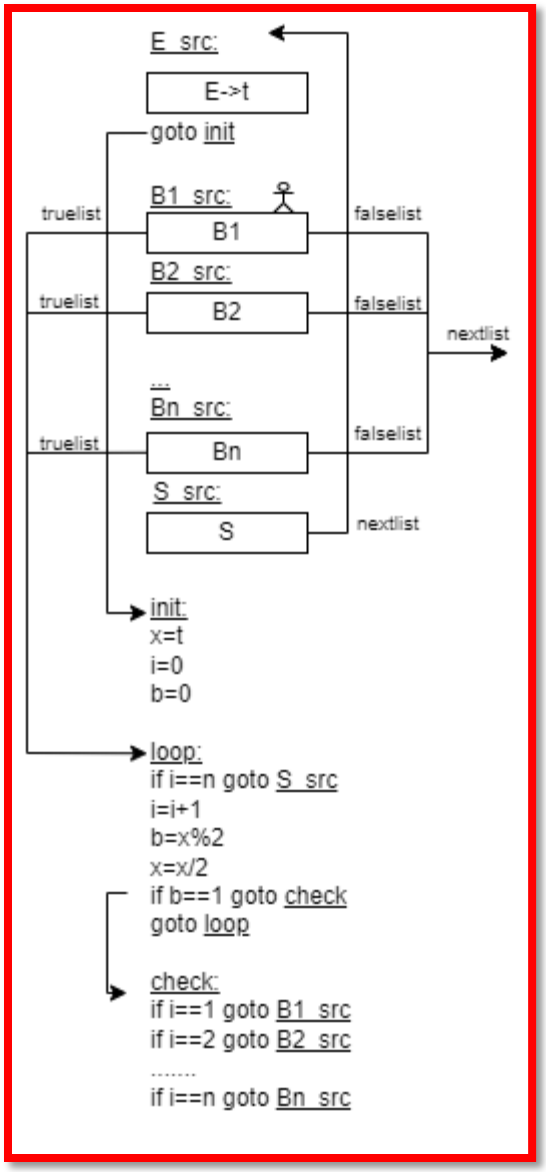
$$S \rightarrow \text{fancy\_while}\ (E; B\_LIST)\ \{\ S\ \}$$
$$B\_LIST \rightarrow B\_LIST\ ,B$$
$$B\_LIST \rightarrow B$$

שימו לב :

- בשפת הביניים קיימות פעולות אריתמטיות עבור חילוק שלמים ומודולו (/ ו-%).
- ניתן להניח שבקוד לא יהיו שגיאות קומפילציה
- אין לשנות את הדקדוק פרט להוספת מרקרים M,N
- ניתן להשתמש במרקרים N,M 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. (6 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.

הפריסה :


E_src:
E->t
goto init
B1_src:
B1
B2_src:
B2
...
Bn_src:
Bn
S_src:
S
truelist
falselist
nextlist
init:
x=t
i=0
b=0
loop:
if i==n goto S_src
i=i+1
b=x%2
x=x/2
if b==1 goto check
goto loop
check:
if i==1 goto B1_src
if i==2 goto B2_src
.....
if i==n goto Bn_src

2. (10 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.

$B\_LIST \rightarrow \mathrm{M}\ B$
$\{\ B\_LIST.quads = makelist(M.quad)$
$B\_LIST.falselist = B.falselist$
$B\_LIST.truelist = B.truelist$
$B\_LIST.len = 1\ \}$

$B\_LIST \rightarrow B\_LIST1\ ,\ M\ B$
$\{\ B\_LIST.quads = merge(B\_LIST.quads, makelist(M.quad))$
$B\_LIST.falselist = merge(B.falselist, B\_LIST1.falselist)$
$B\_LIST.truelist = merge(B.truelist, B\_LIST1.truelist$
$B\_LIST.len = B\_LIST1.len + 1\ \}$

$S \rightarrow \text{fancy\_while}\ (M\ E\ N; B\_LIST)\ \{M1\ S\ \}$
$\{\ bp\big(N.nextlist, nextquad(\,)\big)$
$x = newvar(\,), t = newvar(\,), i = newvar(\,),$
$emit(\ x = E.place)$
$emit(\ i = 0)$
$emit(\ b = 0)$
$bp\big(B\_LIST.truelist, nextquad(\,)\big)$
$loop = nextquad(\,)$
$emit(if\ i == B\_LIST.len\ goto\ M1.quad)$
$emit(i = i + 1)$
$emit(b = x\%2)$
$emit(x = x\%2)$
$hole = makelist(nextquad)$
$emit(if\ b == 1\ goto\ \_\_\ )$
$emit(goto\ ||loop)$
$bp\big(hole, nextquad(\,)\big)$
$for(j = 1; j \leq B\_LIST.len; j++)\{$
$q = B\_LIST.quads.first(\,)$
$emit\big(if\ i == ||\ j\ ||\ goto\ ||\ q\big)\}$
$bp(S.nextlist, M.quad)$

טעויות נפוצות לסעיפים א,ב:

- לחלק ב10 במקום 2
- לשערך את E רק פעם אחת
- לא להתמודד עם המקרה שבו אין ביטים דולקים
- להשתמש בסינטקס לא חוקי בLLVM

משתמשי שפת FanC רוצים שהקוד שלהם יהיה יעיל ככל הניתן. הם שמו לב שרוב השימושים במבנה החדש הינם עם בורר (המשתנה E) קבוע (כלומר, מצויין באופן מפורש בקוד). עלתה ההצעה לשנות את פריסת הקוד מהסעיפים

הקודמים כך שתנאי הלולאה יחוברו באופן קבוע ביניהן לפי הערך הקבוע שהתקבל. לדוגמא עבור הדוגמא בתחילת השאלה שנתיב האמת של התנאי הראשון יפנה ישירות לתחילת התנאי השלישי.

$$S \rightarrow \text{fancy\_while\_fast}\ (\ NUM, B\_LIST\ )\ \{\ S\ \}$$

3. (4 נק') הסבירו במילים אילו שינויים נדרשים לעומת סעיפים א' וב' על מנת לתמוך במבנה היעיל החדש.

כעת שאנחנו יודעים בזמן קומפילציה מהם הביטים הדולקים, כלומר כל ה-Bn-ים שצריך לבדוק ידועים בזמן קומפילציה, ולכן אפשר לחווט את ה-truelist ישירות ל-Bn הבא שנבדק בזמן קומפילציה בלי לעבור דרך הלולאה. אחרי S, אנחנו קופצים חזרה ל-Bn הראשון שנבדק, וגם אין צורך בלולאה וחילוק בזמן ריצה.

טעויות נפוצות:

- לא לתת את הפתרון היעיל ביותר, פתרונות שעדיין דורשים פעולות מתמטיות בזמן ריצה

שאלה 2 : אנליזה סטטית (30 נק')

גבי ודבי, סטודנטים בקורס קומפילציה, מצאו עבודת קיץ בחברה המטפלת במידע רפואי רגיש של משתמשים. עקב דליפת אבטחה חמורה שהייתה לחברה בשנה שעברה, כולם דואגים שמא מידע רגיש נשמר במקומות אסורים. גבי ודבי התנדבו לכתוב אנליזה שמבטיחה כי מידע רגיש **לא נשמר** בשורת קוד מסויימת.

לרוע המזל, גבי ודבי היו עסוקים מאוד בהגשות פרויקטים ובעבודה השוטפת שלהם במשך הקיץ, ולא הגיעו למשימה שהתנדבו אליה. עזרו לגבי ודבי!

א. (3 נק') גבי כתב את שלד הקוד שיבצע chaotic iterations ואליו ישבצו את האנליזה ברגע שימצאו אחת מתאימה. כרגע הקוד מחזיר על כל שורות הקוד בתכנית warning שעשוי להישמר בהן מידע רגיש. גבי טוען שהקוד עונה על הדרישות כפי שהוא ומציע לתת אותו למנהל שלהם. דבי טוענת שחוסר הדיוק של הפתרון יורגש.

1. גבי צודק. באיזה מובן?

גבי צודק מכיוון שהאנליזה בכל זאת נאותה: מכיוון שאף שורה בקוד לא תוחזר על ידי האנליזה כנכונה, אף שורה לא תוחזר באופן שגוי כנכונה. כלומר, כל שורה שהאנליזה לא מזהירה לגביה היא שורה שבוודאות לא שומרת מידע רגיש – באופן ריק.

2. דבי צודקת. באיזה מובן?

דבי צודקת מכיוון שהאנליזה לא תקבל אף תכנית, כולל תכניות שבכלל אין להן שום קשר למידע רגיש.

ב. (12 נק') בתכנה שגבי ודבי רוצים לבדוק יש שני סוגים של מידע רגיש לגבי משתמשים: מידע רפואי וסיסמאות. שניהם נשמרים במשתנים מטיפוס מחרוזת. דבי הגדירה את אוסף הפעולות למחרוזות שהאנליזה תטפל בהן בשלב הניסיוני הראשון:

`"str"`	קבוע מסוג מחרוזת. אינו תלוי במשתמש ולכן אינו מידע רגיש.
`readusername()`	מבקשת מהמשתמש להכניס שם משתמש למערכת ומחזירה את שם המשתמש. שם המשתמש אינו מידע רגיש.
`readpassword()`	מבקשת מהמשתמש להכניס סיסמה למערכת ומחזירה את הסיסמה.
`getbloodtype(username)`	קוראת ממסד נתונים את רשומת המשתמש ומחזירה מחרוזת המציינת את סוג הדם שלו, למשל "O-", "O+", וכו'. סוג דם הוא אינפורמציה סודית רפואית.
`getmedicalhistory(username)`	קוראת ממסד נתונים את רשומת המשתמש ומחזירה מחרוזת המכילה את ההיסטוריה הרפואית של המשתמש. מחרוזת זו יכולה להיות ארוכה מאוד. זוהי אינפורמציה סודית רפואית.
`std::format(format,…)`	מקבלת כפרמטר ראשון מחרוזת פורמט וכפרמטרים נוספים ערכים לשבץ במחרוזת הפורמט, בדומה ל-`printf` אך ללא הדפסה.
`s.substring(n1,n2)`	מחזירה את תת המחרוזת של s מתו מס' n1 עד תו מס' n2 (לא כולל).
`s1 + s2`	מחזירה מחרוזת חדשה המכילה את כל התווים ב-s1 ואחריהם כל התווים ב-s2.
`log(msg)`	מקבלת ביטוי מחרוזת msg ושומרת אותה לקובץ הלוג של המערכת.

ניתן לבצע השמה מכל ביטוי מטיפוס מחרוזת לתוך כל משתנה מסוג `std::string`.

דבי רוצה לבדוק בשלב ראשון כי אף מידע רגיש לא נכתב על ידי פונקציית log. אם ייתכן שנכתב, הכלי של גבי ודבי יחזיר warning עבור השורה. למשל עבור הפונקציה login, הכלי יבדוק את הקריאות ל-log בשורות 9, 18.

```
1: boolean login(int attempts) {
2:   std::string logmsg = "";
3:   while (attempts > 0) {
4:     logmsg += std::format("attempts remaining: {}\n", attempts);
5:     std::string username = readusername();
6:     std::string password = readpassword();
7:     if (authenticate(username,password,attempts - 1)) {
8:       logmsg += std::format("{} authenticated!\n");
9:       log(logmsg);
10:      return true;
11:    }
12:    else {
13:      logmsg += std::format("{} failed to authenticate with {}\n",
14:                            username,password);
15:    }
16:    attempts--;
17:  }
18:  log("Attempts run out!");
19:  return false;
20:}
```

1. הסבירו בקצרה מה נדרש על מנת לבדוק את הקריאות ל-log: מה פריט המידע שהאנליזה צריכה כדי לבדוק את פונקציית log, וכיצד תשתמשו בו?

כדי לבדוק את הקריאות ל-log צריך לבדוק האם המחרוזת המועברת ל-log **עלולה** להכיל מידע רגיש. כלומר, נעקוב אחרי מחרוזות העלולות להכיל מידע רגיש, ובסוף האנליזה נבדוק האם הפרמטר של log עלול להכיל מידע רגיש ואם כן נחזיר warning.

2. לשם הבדיקה, הגדירו את הדומיין האבסטרקטי, את יחס הסדר בדומיין (⊑) ואת פעולת ה-join (⊔). לא נדרש לנסח את α ואת γ.

לדומיין הכי פשוט יש שני איברים, s=may be sensitive, ns=not sensitive, כאשר יחס הסדר הוא $ns \sqsubseteq s$, ופעולת ה-join הנגזרת מיחס הסדר היא $\forall a.\, a \sqcup a = a, ns \sqcup s = s$. חדי הבחנה ישימו לב שזה שקול לגמרי לדומיין חזקה על האיבר הבודד sensitive, כלומר $\mathcal{P}(\{s\})$, עם ⊔=∪ ו-⊑=⊆.

פתרונות רבים כללו גם איבר בקבוצת החזקה ל-not sensitive (לא שגוי, סתם לא נחוץ) או הפרידו בין מידע רפואי לסיסמאות כבר בסעיף הזה. שניהם נכונים כל עוד הבדיקה ב-4 נכונה.

3. הגדירו את הסמנטיקה האבסטרקטית של ביטויים בשפה. הניחו כי אינפורמציה פרטית של משתמשים "מדבקת" גם אם רק חלק ממנה מועתק הלאה. ניתן להניח כי יש ברשותכם את התוצאות של constant propagation על מספרים. אם רלוונטי, תוכלו להניח כי השמה למשתנה כבר מוגדרת כפי שראינו בשיעור.

לא ביקשנו מכם להגדיר את הקריאה או הכתיבה למשתנה בשאלה, אבל לשם השלמות בפתרון נגדיר את $\sigma^{\#}$ להיות פונקציה ממשתנים בסביבה לערכים בדומיין שלנו, וקריאה והשמה יהיו ממומשים כפי שראינו בכיתה. (תזכורת: זה שקול ללהגדיר דומיין מכפלה של k איברים עבור k משתנים, סתם יפה יותר לסמן.)

$[\![\text{"}str\text{"}]\!]^{\#}\sigma^{\#} = ns$, כלומר מחרוזת קבועה אינה תלויה במידע מהמשתמש ולכן בוודאות לא תכיל מידע רגיש.

$[\![readusername(\quad)]\!]^{\#}\sigma^{\#} = ns$, מכיוון ששם משתמש מוגדר כמידע לא רגיש בניגוד לסיסמאות

$[\![readpassword(\quad)]\!]^{\#}\sigma^{\#} = s$, מכיוון שסיסמה היא מידע רגיש

$[\![getbloodtype(username)]\!]^{\#}\sigma^{\#} = s$, מכיוון שסוג דם הוא מידע רפואי ולכן רגיש

$[\![getmedicalhistory(username)]\!]^{\#}\sigma^{\#} = s$, וכך גם היסטוריה רפואית.

חמשת המקרים הללו מהווים את "תנאי העצירה" של החישובים הרקורסיביים בסמנטיקה שלנו. יתר המקרים זקוקים להם על מנת לבצע את החישוב:

$$[\![std :: format(format, a_0, \dots a_k)]\!]^{k}\sigma^{\#} = [\![format]\!]^{\#}\sigma^{\#} \sqcup \bigsqcup [\![a_i]\!]^{\#}\sigma^{\#}$$

כלומר join בין הערך של כל הארגומנטים (מטיפוס מחרוזת) של std::format. מכיוון שה-format string של std::format לא חייב להיות קבוע אלא עשוי להיות משתנה או ביטוי, חייבים לכלול גם אותו.

$$[\![str.substring(n1,n2)]\!]^{\#}\sigma^{\#} = \begin{cases} ns & n1 > n2 \\ [\![str]\!]^{\#}\sigma^{\#} & o.w. \end{cases}$$

בהנתן התוצאות של constant propagation על מספרים, ניתן לדייק את התוצאה של substring ככל שניתן: במידה וה-slice של המחרוזת יצא ריק, אזי מחרוזת ריקה היא בוודאות לא רגישה. מכיוון שרגישות "מדבקת" כל תת מחרוזת לא ריקה של str תהיה במידת הרגישות של str. ניקוד מלא ניתן גם למי שלא דייק את האנליזה ע"י הפרדת המקרה של מחרוזת ריקה אלא הדביק כל תת מחרוזת בערך של str, כל עוד בדומיין $ns \sqsubseteq s$, כי במצב זה מדובר רק באובדן דיוק, כלומר החזרת ערך גבוה יותר בסריג מאשר התוצאה הכי מדוייקת, אבל התוצאה האמיתית של ריצה קונקרטית עדיין תיכלל בריצת האנליזה.

$$[\![s1 + s2]\!]^{\#}\sigma^{\#} = [\![s1]\!]^{\#}\sigma^{\#} \sqcup [\![s2]\!]^{\#}\sigma^{\#}$$

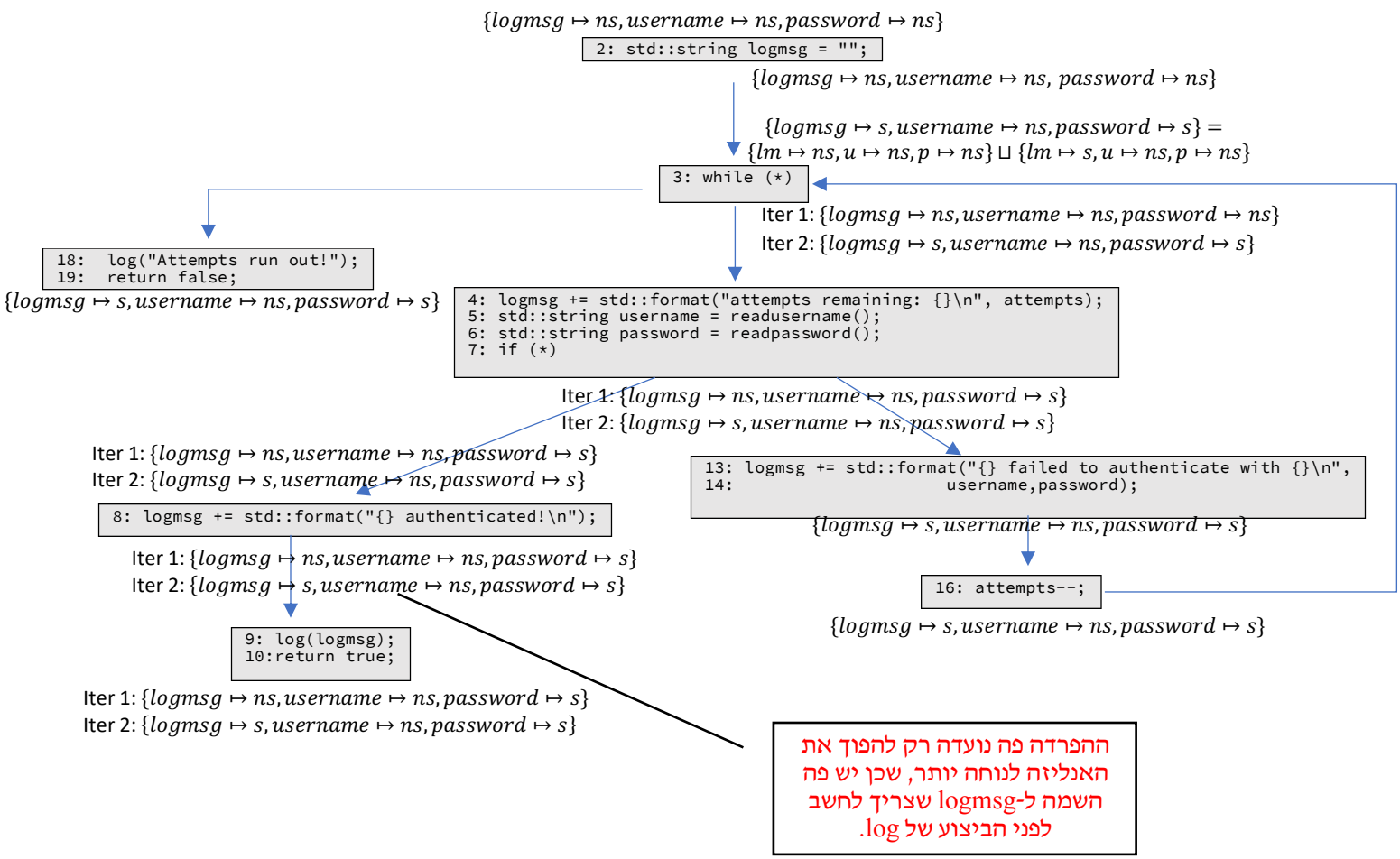
לבסוף, נותרה הפונקציה log. מכיוון ש-log לא מחזירה ערך, אין חשיבות לערך שלה עבור האנליזה. לפיכך ניתן להחזיר פה כל ערך וכל התשובות התקבלו **כל עוד** הן לא "זרקו שגיאה" כחלק מהסמנטיקה של log – אם הסמנטיקה של log תעצור את האנליזה, אז אנחנו מפריעים לה להתכנס ובמקרה של מספר פונקציות log עשויים לעצור כאשר הערכים עבור פונקציית log אחרת עדיין אינם סופיים. למשל, לו log בשורה 18 הייתה גם log(logmsg), אז לעצור את האנליזה על ה-warning בשורה 9 היה יכול לגרום לנו לפספס את שורה 18 שמובטח לנו שהתעדכנה בערכים מהלולאה רק כאשר האנליזה מתכנסת.

4. הסבירו את הבדיקה שמתבצעת עבור כל קריאה לפונקציה `log` כדי לסיים את האנליזה.

אחרי שהאנליזה התכנסה, נחשב עבור כל קריאה ל-log את $[\![msg]\!]^{\#}\sigma^{\#}$. אם $[\![msg]\!]^{\#}\sigma^{\#} = ns$ אזי המידע בוודאות לא רגיש ולא נתריע. אחרת, תהיה warning עבור הקריאה.

ג. (10 נק') הריצו את האנליזה

1. ציירו את ה-CFG של הפונקציה `login`
2. השתמשו באנליזה שלכם כדי לנתח את הפונקציה. ניתן להתעלם מתנאים מספריים, ומתנאים המכילים את הפונקציה `authenticate`.



3. ציינו את מספרי השורות בהן תוחזר ה-warning וכיצד רואים זאת לפי מה שהגדרתם בתת סעיף ב.4.

נשתמש בערכי האנליזה כדי לבדוק את שתי הקריאות ל-log, בשורה 9 ובשורה 18. בכניסה לשניהם ערכי המשתנים האבסטרקטיים בתום האנליזה הם $\{logmsg \mapsto s, username \mapsto ns, password \mapsto s\}$. בשורה 9 הארגומנט ל-log הוא logmsg, $[\![logmsg]\!]^{\#}\sigma^{\#} = \sigma^{\#}(logmsg) = s \neq ns$ ולכן לפי החישוב מ-ב.4 יהיה warning.

לעומת זאת, בשורה 18, $[\![\text{"}Attempts\ run\ out\text{"}]\!]^{\#}\sigma^{\#} = ns$ ולכן לפי החישוב לא יהיה warning.

ד. (5 נק') דבי מצא במדריך החברה פסקה שערורייתית הגורסת כי אין בעיה לכתוב ל-`std::cout` מחרוזות המכילות סיסמאות, אך אסור לכתוב לשם מידע רפואי סודי. ל-`log` אסור לכתוב גם את אלה וגם את אלה. האם נדרש שינוי בדומיין שלכם כדי שתתקבל שלא תופיע warning בשורה 190 כאן?

```
189:     std::string password = readpassword();
190:     std::cout << password << std::endl;
```

אם לא – הדגימו מה יקרה בשורות 189-190 באנליזה שלכם.

עבור סעיף ב' הכולל הפרדה בין סיסמאות למידע רפואי, תשובה מלאה נראית כך:

לא, אין צורך לשנות את הדומיין שכן הוא כבר כולל הפרדה בין מידע רפואי לסיסמאות. בשורה 189 המשתנה password יקבל את הערך {p}, ובשורה 190 כאשר הערך המועבר ל-std::cout ייבדק האנליזה תשערך אותו כ-{p}, ולפי הבדיקה החדשה הנדרשת ל-std::cout שתבדוק האם $m \in [\![str]\!]^{\#}[\![msg]\!]^{\#}\sigma^{\#}$ לא תהיה warning.

אם כן – תקנו את הדומיין שלכם: תארו את איברי הדומיין ויחס הסדר לאחר השינוי, והסבירו <u>במילים ובקצרה</u> כיצד הסמנטיקה של ביטויים תשתנה.

עבור סעיף ב' שכלל רק התייחסות למידע "רגיש", נשנה את הדומיין כדי להפריד בין מידע רפואי וסיסמאות. דומיין שהוגדר מראש כקבוצת חזקה יוחלף ב-$\mathcal{P}(m,p)$ עם אותו יחס סדר ואותו join. הסמנטיקה של פונקציות המחזירות מידע רגיש תחזיר כעת את הסוג הספציפי של מידע רגיש המוחזר, והבדיקה עבור cout תיתן warning רק אם m כלול בקבוצה.

עבור דומיין שהוגדר עם איברים קונקרטיים ניתן להוסיף רק איבר אחד חדש, p, כך:

$$\begin{array}{c} s \\ | \\ p \\ | \\ ns \end{array}$$

כאן, הסמנטיקה של פונקציות המחזירות מידע רפואי לא תשתנה והן עדיין יחזירו s (כלומר, עשוי להכיל את הדרגה הכי חמורה של מידע רגיש), והסמנטיקה של readpassword בלבד תשתנה כדי להחזיר p (עשוי להכיל מידע רגיש אבל רק מסוג סיסמה). כך הבדיקה עבור cout : std: תוגדר כי יהיה warning רק אם הערך האבסטרקטי של הפרמטר הוא s.

בהצלחה!

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):

$$\begin{cases} \\ \\ \\ \end{cases}$$

M[A , t] =	$A \rightarrow \alpha$	$t \in select(A \rightarrow \alpha)$
	error	$t \notin select(A \rightarrow \alpha)$ for all $A \rightarrow \alpha \in P$

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: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
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. $Statement \rightarrow AUTO\ ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
19. $Statement \rightarrow Call\ SC$
20. $Statement \rightarrow RETURN\ SC$
21. $Statement \rightarrow RETURN\ Exp\ SC$
22. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
23. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
24. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
25. $Statement \rightarrow BREAK\ SC$
26. $Statement \rightarrow CONTINUE\ SC$
27. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
28. $Call \rightarrow ID\ LPAREN\ RPAREN$
29. $ExpList \rightarrow Exp$
30. $ExpList \rightarrow Exp\ COMMA\ ExpList$
31. $Type \rightarrow INT$
32. $Type \rightarrow BYTE$
33. $Type \rightarrow BOOL$
34. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
35. $Exp \rightarrow Exp\ BINOP\ Exp$
36. $Exp \rightarrow ID$
37. $Exp \rightarrow Call$
38. $Exp \rightarrow NUM$
39. $Exp \rightarrow NUM\ B$
40. $Exp \rightarrow STRING$
41. $Exp \rightarrow TRUE$
42. $Exp \rightarrow FALSE$
43. $Exp \rightarrow NOT\ Exp$
44. $Exp \rightarrow Exp\ AND\ Exp$
45. $Exp \rightarrow Exp\ OR\ Exp$
46. $Exp \rightarrow Exp\ RELOP\ Exp$
47. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$