פתרון מבחן סוף סמסטר – מועד ב'

זהו הפתרון עבור טור א'

בעמוד 17 מופיעות התשובות הנכונות עבור טורים ב' ו-ג'.

לשאלה 14 שתי תשובות נכונות

מרצה אחראי: ד"ר איל זקס

מתרגלים: מתן פלד, אורן בניטה בן שמחון, שי גנדלמן, נדב רובינשטיין

הוראות:

א. בטופס המבחן 11 עמודים, וכן 6 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
ב. משך המבחן **שלוש שעות (180 דקות).**
ג. אסור כל חומר עזר חיצוני.
ד. במבחן 20 שאלות. כל השאלות הינן חובה. כל שאלה בת 5 נקודות.
ה. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.
ו. אין צורך להגיש את טופס מבחן זה בתום הבחינה.
ז. את התשובות לשאלות יש לסמן בטופס התשובות הנפרד בלבד.

טור א

בהצלחה!

Backpatching

נרצה להוסיף לדקדוק מבנה בקרה של לולאה עם מספר איטרציות שמחושב לפני הכניסה ללולאה, במקום תנאי עצירה כללי שיכול להשתנות בכל איטרציה. מבנה הבקרה מחזיר חיווי אם התבצעה לפחות איטרציה אחת.
לדוגמא:

```
if (do_loop (count * 4) {
   read(&buf);
   print(buf);
 })
 print ("loop was executed at-least once");
```

בדוגמא הנ"ל בתחילת הלולאה מחשבים את מספר האיטרציות; אם מספר זה הוא שלילי או אפס, אין לבצע את תוכן הלולאה כלל, ויש להחזיר false. אחרת יש למבצע את תוכן הלולאה כמספר שחושב, ולהחזיר true. מספר האיטרציות **לא** יושפע מחישובים בתוך הלולאה, לדוגמא אם חישובים אלה ישנו את ערכו של משתנה count.

נוסיף לדקדוק את הכלל הבא עבור מבנה בקרה זה:

B → do_loop LPAREN E RPAREN S;

שימו לב:

- בדקדוק אותו אנו מרחיבים יש למשתנים S, E ו-B כללי גזירה נוספים, ואין break.
- למשתנה S יש תכונת nextlist ולמשתנה B תכונות truelist ו-falselist במסגרת הטלאה לאחור של חישוב מקוצר - short-circuit evaluation, backpatching.
- ניתן להניח שהקוד של משתנה S תמיד מסתיים ב-goto, כולל עבור משתנים S הנגזרים לפעולות read ו-print כבדוגמא לעיל.
- 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.
- יש לייצר קוד ביניים מסוג קוד שלשות כמופיע בדפי הנוסחאות המצ"ב.

שאלה מספר 1:

היכן יש למקם מרקר M ומרקר N בכלל הגזירה?

א. B → do_loop LPAREN E M N RPAREN S;

ב. B → do_loop LPAREN E N M RPAREN S;

ג. B → do_loop LPAREN M E N RPAREN S;

ד. B → do_loop LPAREN N E M RPAREN S;

פתרון שאלה מספר 1:

יש למקם N בין E ל-S, אחרת תמיד נבצע את S מיד לאחר ביצוע E, גם כאשר E.place שלילי.
יש למקם M מיד לפני S, כלומר מיד לפני RPAREN, על מנת שנוכל לקפוץ לבצע את S מבלי לבצע את E לפניכן.

לאחר שהוספנו לכלל הגזירה את המרקרים ואת הפעולות הסמנטיות שמייצרות את קוד הביניים, התערבבו לנו הפעולות ויש צורך לסדר אותן מחדש.
ידוע שישנן סה"כ 11 פעולות סמנטיות, מתוך אוסף הפעולות הבא:

1. backpatch(N.nextlist, nextquad);	9. emit "goto ___";
2. backpatch(N.nextlist, M.quad);	10. emit "goto M.quad";
3. backpatch(S.nextlist, nextquad);	11. v = newtemp();
4. backpatch(S.nextlist, M.quad);	12. emit "v = v - 1";
5. B.falselist = nextquad;	13. emit "v = E.place";
6. B.truelist = nextquad;	14. emit "v = 0";
7. emit "if E.place <= 0 goto ___";	15. emit "if v > 0 goto ___";
8. emit "if E.place > 0 goto ___";	16. emit "if v > 0 goto M.quad";

שאלה מספר 2:

מהן שלוש הפעולות הראשונות בכלל הסמנטי, לפי סדר משמאל לימין?

א. 3,5,8
ב. 4,6,7
ג. 1,5,7
ד. 2,6,8

פתרון שאלה מספר 2:

כל 16 הפעולות הנ"ל עושות שימוש רק - N אחד וב-M אחד, והללו חייבים להיות ממוקמים בהתאם לשאלה 1.
בכל 4 התשובות, 3 הפעולות הראשונות הינן : backpatch, השמה ל-B.*list, ויצירת קפיצה מותנית ע"פ E.place.
אין לבצע backpatch שמחווט את N ל-M (פעולה 2 : מייתרת את הצורך ב-N) או את S ל-M (פעולה 4 : תיווצר לולאה אינסופית), ואין לחווט את S ל-nextquad (פעולה 3 : תבצע קפיצה ע"פ E.place אחרי כל ביצוע של S ; יש לקרוא את ערכו של E.place רק לפני ביצוע S). מכאן שיש לחווט את N ל-nextquad (פעולה 1), והאפשרות היחידה שנותרה מכתיבה השמה ל-B.falselist (פעולה 5) וקפיצה מותנית ע"פ E.place <= 0 ליציאה (פעולה 7).

שאלה מספר 3:

מהן שלוש הפעולות הבאות בכלל הסמנטי, לפי סדר משמאל לימין?

א. 11,13,10
ב. 11,14,10
ג. 11,13,9
ד. 11,12,9

פתרון שאלה מספר 3:

בכל מקרה תחילה נייצר משתנה חדש ע"י newtemp() (פעולה 11). מיד לאחר מכן אין לייצור פעולת v=v-1 בטרם הושם למשתנה v ערך כלשהו (פעולה 12 ; פעולת newtemp רק מייצרת שם משתנה חדש, ללא יצירת כל פעולת איתחול המשימה למשתנה זה ערך). לאחר newtemp יש לייצר פעולה המאתחלת את v לאפס או ל-E.place. היות והפעולות היחידות לעדכון והשוואת v מאפשרות הפחתה ב-1 והשוואה לאפס, יש לאתחל את v ל-E.place (פעולה 13) ולא לאפס (פעולה 14). אין לייצר קפיצה לכתובת חסרה ללא שמירת הפעולה ברשימה כלשהיא לשם ביצוע backpatching מאוחר יותר (פעולה 9). נשארה האפשרות לקפוץ ל-M.quad (פעולה 10). ואכן בשלב זה ידוע לנו שיש לבצע לפחות איטרציה אחת, וניתן לקפוץ ישירות ל-S ולבצע אותה.

שאלה מספר 4:

מהן שתי הפעולות הבאות בכלל הסמנטי, לפי סדר משמאל לימין?

א. 4,12
ב. 4,14
ג. 3,12
ד. 3,13

פתרון שאלה מספר 4:

ברור שאין לחווט את S ל-M.quad שכן הדבר ייצור לולאה אינסופית (פעולה 4, ראינו גם בשאלה 2). נותר לחווט את S ל-nextquad (פעולה 3), ולאחר מכן יש לטפל ביצירת קוד שירוץ לאחר כל איטרציה/S ; זוהי פעולת ההפחתה ב-1 מ-v (פעולה 12), ולא אתחול של v (פעולה 13 ; אין לקרוא את ערכו של E.place רק לפני ביצוע S).

שאלה מספר 5:

מהן שלוש הפעולות האחרונות בכלל הסמנטי, לפי סדר משמאל לימין?

א. 6,15,9
ב. 16,6,9
ג. 16,6,10
ד. 15,5,10

פתרון שאלה מספר 5:

אין לייצר קפיצה לכתובת חסרה ללא שמירת הפעולה ברשימה כלשהיא לשם ביצוע backpatching מאוחר יותר (פעולה 15). קפיצה ל-B.truelist כאשר v>0 לצא מהלולאה בטרם בוצעו כל האיטרציות (פעולות 6,15). נותרה האפשרות לקפוץ לאיטרציה נוספת, כלומר ל-M.quad, אם v חיובי (פעולה 16), ואז (אחרת, אם v התאפס) לרשום את הפעולה הבאה ב-B.truelist (פעולה 6). כל שנותר הוא לייצר קפיצה ריקה, שתצא מהלולאה ל-B.truelist (פעולה 9).

אופטימיזציה

לפונקציה "void func(z)" אשר מקבלת את המשתנה z כפרמטר מוגדר קוד הביניים הבא:

```
entry:      y = 2
            p = z * y
            x = z << 1
            L = y
            goto next
            p = 3
next:       q = p * L
            print p
            print q
            r = p + x
            print r
            return
```

זהו כל קוד הביניים של הפונקציה; כל המשתנים המוגדרים בפונקציה הם משתנים לוקליים; קוד מחוץ לפונקציה יכול לקפוץ רק לתחילתה (ל- entry).

המעבד עבורו מקמפלים את התוכנית מעדיף פעולות לוגיות – (bit logic: &,|,<<,>>) על פני פעולות אריתמטיות רגילות – (+,-,*,/).

שאלה מספר 6:

עליך להריץ את האופטימיזציות השונות שנלמדו עד קבלת קוד יעיל ביותר ולציין איזו אופטימזיה **אינה** מתבצעת?

א. Constant propagation

ב. Useless code elimination

ג. Common sub expression elimination

ד. Constant folding

ה. Unreachable code elimination

ו. Algebraic simplification

פתרון שאלה מספר 6:

א. CP: Constant propagation

ב. UsCE: Useless code elimination

ג. CSE: Common sub expression elimination

ד. Constant folding

ה. UnCE: Unreachable code elimination

ו. AS: Algebraic simplification

```
entry:    ~~y~~ = 2                                CP
          p = z * y = z * 2 = z << 1               CP + AS
          ~~x~~ = z << 1 = p                       CSE + UsCE
          ~~L~~ = y = 2                            CP + UsCE
          ~~goto next~~                            Branch elimination
          ~~p~~ = 3                                UnCE
next:     q = p * L = p * 2 = p << 1               CP + AS
          print p
          print q
          ~~r~~ = p + x = p+p = 2*p = p<<1 = q     Copy prop. + AS + CSE DCE
          print ~~r~~ q                            Copy prop.
          return
```

שאלה מספר 7:

כמה בלוקים בסיסיים וכמה פקודות נותרו לאחר הרצת האופטימיזציות עד לקבלת הקוד היעיל ביותר?
יש לספור את return כפקודה אחת.

א. בלוק אחד ו-3 פקודות

ב. 2 בלוקים ו-7 פקודות

ג. 3 בלוקים ו-8 פקודות

ד. בלוק אחד ו-6 פקודות

ה. בלוק אחד ו-7 פקודות

ו. 3 בלוקים ו-10 פקודות

פתרון שאלה מספר 7:

נשאר עם בלוק אחד ובו 6 פקודות:

```
p = z << 1
q = p << 1 (= z << 2)
print p
print q
print q
return
```

עבור שלוש השאלות הבאות, נוספה לשפת הביניים האפשרות להגדיר משתנה כמשתנה גלובלי "**נדיף**" -volatile.
משמעות הגדרה זו היא שכל פניה למשתנה נדיף (קריאה או כתיבה) נדרשת לגשת לזכרון, שכן ערכו של המשתנה יכול להקרא ו/או להכתב לזכרון במקביל לריצת הקוד המתקמפל.

דוגמא למשתנה נדיף: משתנה גלובלי המאפשר עדכון וסנכרון בין תהליכים שונים הרצים במקביל.

שאלה מספר 8:

כעת נתון שמשתנה **L** בפונקציה func המופיעה לעיל הוגדר כמשתנה גלובלי נדיף, במקום משתנה לוקלי. עליך להריץ שוב את כל האופטימיזציות האפשרויות על מנת לקבל קוד יעיל ביותר, אשר שומר על ההתנהגות של משתנה L כמשתנה נדיף. כמה פקודות נותרו לאחר הרצת האופטימיזציות?
יש לספור את return כפקודה אחת.

א. 8

ב. 7

ג. 6

ד. 9

ה. 3

ו. 4

פתרון שאלה מספר 8:

```
entry:      ~~y~~ = 2
            p = z * y = z * 2 = z << 1
            ~~x~~ = z << 1 = p
            L = y = 2                       Keep store to L
            ~~goto next~~                   Branch elimination
            ~~p~~ = 3
next:       q = p * L                       Keep load from L
            print p
            print q
            ~~r~~ = p + x = p+p = 2*p = p<<1      No longer == q
            print r
            return
```

נשאר עם בלוק אחד ובו 8 פקודות:

```
p = z << 1
L = 2
q = p  * L
print p
print q
r = p << 1 (= z << 2)
print r
return
```

שאלה מספר 9:

אילו מהאופטימיזציות הבאות **יכולות** להיות מושפעות אם נשנה את ההגדרה של משתנים לנדיפים?

א. Unreachable code elimination

ב. Useless code elimination

ג. תשובות ב' ו-ו' נכונות

ד. תשובות ב' ו-א' נכונות

ה. Constant folding

ו. Common sub expression elimination

פתרון שאלה מספר 9:

קוד שלא ניתן להגיע אליו, ניתן להסרה, ללא קשר למשתנים כלשהם, כולל נדיפים, לכן unreachable code elimination **איננו** מושפע.

אין למחוק פקודה שכותבת למשתנה נדיף גם כאשר אין לה כל שימוש בהמשך, לכן useless code elimination יכול להיות **מושפע**. זאת משום שלמעשה יכול להיות לה שימוש בהמשך, אותו האנליזה/אופטימיזציה לא רואה; הפקודה לא בהכרח useless.

פעולה על ערכים קבועים ניתנת לביצוע בזמן קומפילציה ללא קשר למשתנים כלשהם, כולל נדיפים, לכן constant folding **איננו** מושפע.

ביטוי המופיע פעמיים המכיל משתנה נדיף מצריך קריאת המשתנה פעמיים, לכן CSE יכול להיות **מושפע**. זאת משום שערכו של המשתנה יכול להשתנות בין שני המופעים, למרות שהאלניזה/אופטימיזציה לא רואה זאת. הביטויים אינם בהכרח common.

כעת נגדיר את האופטמיזציה "**Shift Combine**" הפועלת על זוגות של פעולות i1,i2 מהצורה:

```
(i1:) a = b << c
(i2:) z = a << d
```

כאשר i1 הינה ההגדרה היחידה של משתנה a המגיעה ל-i2.
האופטימיזציה מחליפה את פעולה i2 בשתי פעולות i3,i4 כאשר t משתנה זמני חדש:

```
(i3:) t = c + d
(i4:) z = b << t
```

<u>שאלה מספר 10:</u>

אילו מהטענות הבאות נכונות?

א. מותר לאופטימיזציית shift combine לפעול כאשר משתנה a (ב-i1,i2 לעיל) הינו משתנה נדיף.

ב. אופטימיזצית shift combine יכולה לשנות את קוד הביניים של פונקצית func לעיל עם L כמשתנה נדיף, אם תופעל לאחר הפעלת אופטימיזציות אחרות.

ג. הפעלת אופטימיזצית shift combine עלולה ליצור קוד פחות יעיל.

ד. תשובות ב', ג' ו-ה' נכונות.

ה. הפעלת אופטימיזצית shift combine עשויה ליצור קוד יותר יעיל בצירוף אופטימיזציות נוספות, כאשר חלק מהמשתנים הינם קבועים ול-a יש שימוש יחיד (בפקודה i2).

ו. תשובות ג' ו-ה' נכונות.

<u>פתרון שאלה מספר 10:</u>
חובה להמשיך לכתוב ולקרוא מ-a אם הוא נדיף.

פתרון שאלה 8 מראה כיצד ניתן להפעיל האופטימיזציה על פונ' func לאחר ריצת אופטימיזציות אחרות, גם כאשר L משתנה נדיף.

האופטימיזציה עלולה להוסיף פעולת חיבור שהיא יקרה יותר מפעולה בינארית.

אם המחוברים קבועים (סכום ערכי ה-shift), כמו במקרה של func, ניתן לקפל את חישוב סכומם, ולחסוך פעולת חיבור יקרה; אם בנוסף ל-a שימוש יחיד ניתן יהיה למחוק את הפעולה שמגדירה אותו, היות והאופטימיזציה הופכת אותו לחסר שימוש.

Data Flow Analysis

בשאלות הבאות נעסוק באנליזות שונות מסוג Data-Flow Analysis אשר חושבו **עבור תחילת וסוף כל בלוק בסיסי** עבור ייצוג ביניים מסוג קוד שלשלות של פונקציה כלשהי. נתעניין מתי התוצאות של אנליזה עשויות להשנות כאשר נעשית פעולה נתונה בייצוג הביניים; כלומר מתי הרצת האנליזה בשנית לאחר ביצוע הפעולה עשויה לייצר תוצאה שונה מזו שנוצרה תחילה.

שאלה מספר 11:

לאחר שחישבנו **משתנים שערכם תמיד זוגי או ערכם תמיד אי-זוגי** בתחילת וסוף כל בלוק בסיסי, אילו מהפעולות הבאות עלולות לשנות את תוצאת האנליזה?

א. הוספת פקודה x = y << 1 כפקודה אחרונה בבלוק בסיסי אשר בסופו לא היה ידוע אם x זוגי או אי-זוגי

ב. הסרת פקודה x = x + 1 שנמצאת בסוף הבלוק הבסיסי שלה, אשר בסופו לא היה ידוע אם x זוגי או אי-זוגי

ג. טענות ב' ו-ה נכונות

ד. אף טענה איננה נכונה

ה. הסרת פקודה print z מסוף בלוק בסיסי אשר בתחילתו היה ידוע ש-z זוגי

פתרון שאלה מספר 11:

x = y << 1 מגדירה את ערכו של x להיות זוגי, והדבר נכון לסוף הבלוק שם דבר זה לא היה ידוע.
x = x + 1 הופכת זוגיות, אך אם לא היתה ידועה היא תשאר לא ידועה.
print איננה משפיעה על ערכים כלל, ובכלל זה על זוגיותם.

שאלה מספר 12:

לאחר שחישבנו אילו **משתנים חיים** בתחילת וסוף כל בלוק בסיסי, אילו מהפעולות הבאות עלולות לשנות את תוצאת האנליזה?

א. הסרת פקודה print z מסוף בלוק בסיסי אשר בתחילתו היה ידוע ש-z איננו חי

ב. הסרת פקודה x = x + 2 שנמצאת בסוף הבלוק הבסיסי שלה, אשר בתחילתו היה ידוע ש-x איננו חי

ג. טענות ב' ו-א' נכונות

ד. הוספת פקודה print y כפקודה הראשונה בבלוק בסיסי אשר בתחילתו היה ידוע ש-y איננו חי

ה. אף טענה איננה נכונה

פתרון שאלה מספר 12:

הסרת פקודה המשתמשת ב-x או ב-z עלולה להפוך אותו למת בתחילת הבלוק, אך הוא כבר ידוע שם כמת (הפקודות שהוסרו השתמשו בערכים שהוגדרו ע"י פקודות **בתוך** הבלוק).
הוספת print y גורמת ל-y להיות חי בתחילת הבלוק, מקום בו y לא היה חי.

שאלה מספר 13:

לאחר שחישבנו את **ההגדרות המגיעות** reaching definitions לתחילת וסוף כל בלוק בסיסי, אילו מהפעולות הבאות עלולות לשנות את תוצאת האנליזה?

א. אף טענה איננה נכונה

ב. הוספת פקודה print y כפקודה הראשונה בבלוק בסיסי אשר אל תחילתו מגיעות מספר הגדרות של משתנה y

ג. טענות ב' ו-ה' נכונות

ד. הסרת פקודה x = x + 0 שנמצאת בסוף הבלוק הבסיסי שלה, כלומר היא היתה הפקודה האחרונה בבלוק

ה. הסרת פקודה print z שנמצאת בסוף בלוק בסיסי אשר אל סופו מגיעה הגדרה יחידה של משתנה z

פתרון שאלה מספר 13:

print איננה מגדירה כל משתנה על כן הוספתה או הסרתה אינן משפיעות על הגדרות מגיעות.
הפעולה x=x+0 מגדירה את x, והגדרה זו מגיעה לסוף הבלוק בהיותה הפקודה האחרונה, לכן הסרתה תשנה את ההגדרות של x המגיעות אל סוף בלוק זה.

שאלה מספר 14:

לאחר שחישבנו אילו **משתנים חיים** בתחילת וסוף כל בלוק בסיסי, אילו מהפעולות הבאות עלולות לשנות את תוצאת האנליזה?

א. הסרת פקודה x = x >> 2 שנמצאת בסוף הבלוק הבסיסי שלה, אשר בתחילתו היה ידוע ש-x חי

ב. הוספת פקודה y = 0 כפקודה הראשונה בבלוק בסיסי אשר בסופו היה ידוע ש-y איננו חי

ג. טענות ב'ו ד' נכונות

ד. הוספת פקודה print z בסוף בלוק בסיסי אשר בתחילתו היה ידוע ש-z חי

ה. אף טענה איננה נכונה

פתרון שאלה מספר 14:

יתכן והשימוש של x ע"י x = x >> 2 היא הסיבה היחידה ש-x חי בתחילת הבלוק, ואז הסרתו תהפוך את x למת בתחילת הבלוק.
הוספת פעולה y = 0 בתחילת בלוק תגרום לכך שמשתנה y לא יהיה חי בתחילת הבלוק, יתכן ותחילה y היה חי בתחילת בלוק.
הוספת פעולה כמו print z עשויה לגרום ל-z להיות חי בתחילת הבלוק, אך הוא כבר ידוע כחי שם (בשל שימוש אחר/נוסף).

שאלה מספר 15:

לאחר שחישבנו **משתנים שערכם תמיד זוגי או ערכם תמיד אי-זוגי** בתחילת וסוף כל בלוק בסיסי, אילו מהפעולות הבאות עלולות לשנות את תוצאת האנליזה?

א. אף טענה איננה נכונה

ב. הוספת פקודה print y כפקודה הראשונה בבלוק בסיסי אשר בתחילתו היה ידוע שערכו של y זוגי

ג. הסרת פקודה print z מסוף בלוק בסיסי אשר בסופו היה ידוע שערכו של z אי-זוגי

ד. הסרת פקודה x = x + 2 שנמצאת בסוף הבלוק הבסיסי שלה, כלומר היא היתה הפקודה האחרונה בבלוק

ה. טענות ד' ו-ב' נכונות

פתרון שאלה מספר 15:

print איננה מגדירה כל משתנה על כן הוספתה או הסרתה אינן משפיעות על ערכי משתנים הידועים כאי/זוגיים.
הפקודה x = x + 2 שומרת על הזוגיות של x, בין אם זו ידועה ובין אם לאו.

בשאלות הבאות, אותיות גדולות A,B,... הם משתנים, S הוא המשתנה ההתחלתי, ואותיות קטנות a,b,... הם טרמינלים.

שאלה מספר 16:

נתון הדקדוק הבא G1 שאינו ב-$LL(1)$:

(G1)

$S \to AB$

$A \to a \mid aA$

$B \to bB \mid bC$

$C \to aC \mid \epsilon$

ונתונים שלושה דקדוקים נוספים G2, G3, G4:

(G2)	(G3)	(G4)
$S \to aA$	$S \to aAbBA$	$S \to BA$
$A \to aA \mid bB$	$A \to aA \mid \epsilon$	$A \to aA \mid \epsilon$
$B \to bB \mid aC$	$B \to bB \mid \epsilon$	$B \to aAC$
$C \to aC \mid \epsilon$		$C \to b \mid bC$

אילו מהדקדוקים הבאים נמצאים ב-$LL(1)$ וגם **שקולים** לדקדוק G1, כלומר שני הדקדוקים מגדירים את אותה שפה?

א. אף אחת מהתשובות אינה נכונה

ב. G2

ג. G4

ד. G3

פתרון שאלה מספר 16:

השפה של הדקדוק היא $a^+b^+a^*$.

השפה של G2 היא $a^+b^+a^+$, והוא ב-LL(1).

השפה של G3 היא גם $a^+b^+a^*$ וניתן לראות שהוא ב-LL(1).

השפה של G4 היא גם $a^+b^+a^*$ אבל יש קונפליקט עבור משתנה C.

שאלה מספר 17:

נתונים ארבעת הדקדוקים הבאים G5, G6, G7, G8:

(G5)	(G6)	(G7)	(G8)
$S \to A \mid B$	$S \to aA \mid bB$	$S \to AB \mid BA$	$S \to A \mid B$
$A \to cA \mid aAa \mid d$	$A \to cBB \mid a$	$A \to aAB \mid aBA \mid c$	$A \to aaA \mid aA \mid a$
$B \to bB \mid cBc \mid \epsilon$	$B \to A \mid b$	$B \to b$	$B \to bB \mid b$

איזה דקדוק נמצא ב-$LR(0)$ אבל אינו ב-$LL(1)$?

א. G5

ב. G6

ג. G7

ד. G8

פתרון שאלה מספר 17:

G5 לא נמצא באף אחת מהמחלקות: עבור משתנה S יש קונפליקט LL(1), וכבר במצב הראשון של LR(0) מקבלים קונפליקט shift/reduce.

G6 נמצא גם ב- LL(1) וגם ב-LR(0).

G7 לא נמצא ב-LL(1) כי יש קונפליקט עבור A, אבל כן ב-LR(0). **תשובה נכונה.**

G8 הדקדוק לא חד-משמעי, ולכן לא ב-LR(0) ולא ב-LL(1).

שאלה מספר 18:

נתון הדקדוק G הבא, המאפשר גזירה של רשימות של מספרים ורשימות מקוננות:

דקדוק G

$S \rightarrow L$

$L \rightarrow [\ ] \mid [\ N\ ]$

$N \rightarrow E \mid E; N$

$E \rightarrow L \mid num$

כאשר S, N, L, E משתנים ו-num,[,],; טרמינלים, ו-S הוא המשתנה ההתחלתי.

אילו קונפליקטים יש בדקדוק הנ"ל ביחס למנתח $\boldsymbol{SLR}$?

א. reduce/reduce

ב. shift/reduce וגם reduce/reduce

ג. אין קונפליקט

ד. shift/reduce

פתרון שאלה מספר 18:

לדקדוק אכן יש קונפליקט shift/reduce ב-LR(0), אבל ב-SLR הוא נפתר כי ה-follow פותר את הקונפליקט, ולכן אין קונפליקטי SLR.

שאלה מספר 19:

מהי הקבוצה **הקטנה ביותר** בה נמצא **הדקדוק** הבא?

$S \rightarrow T$

$T \rightarrow Ab \mid bAc \mid ac$

$A \rightarrow a$

א. $LR(0)$

ב. $LALR$

ג. SLR

ד. $LR(1)$

<u>הערה</u> - **קטנה** ביחס להכלה של קבוצות דקדוקים, נזכיר ש-$LR(0) \subset SLR \subset LALR \subset LR(1)$, לכן אם דקדוק נמצא ב-$LR(0)$ אז $LR(0)$ היא הקטנה ביותר, אם ב-SLR ולא ב-$LR(0)$ אז SLR היא הקטנה ביותר, וכו'...

<u>תזכורת</u> – מנתח SLR מתקבל על ידי בניית מנתח $LR(0)$ לדקדוק, והוספת פעולת $Reduce$ למשתנה X בעמודה a בטבלת הפעולות אם ורק אם a נמצא ב-$FOLLOW(X)$.

מנתח $LALR$ מתקבל על ידי בניית מנתח $LR(1)$ לדקדוק, ואיחוד כל זוג מצבים באוטומט שקבוצת פריטי ה-$LR(1)$ בהם זהה עד כדי lookahead שונה. לבסוף נוסיף פעולת $Reduce$ למשתנה X בעמודה a בטבלת הפעולות ע"פ פריטי ה-$LR(1)$ במצבי האוטומט המאוחדים.

<u>פתרון שאלה מספר 19:</u>

הדקדוק נמצא ב-*LALR*, ניתן לראות זאת על ידי בניית מנתח *LR(1)* לדקדוק, איחוד מצבים דומים (אין כאלה), ובדיקה שאין קונפליקטים. האינטואיציה לגבי למה הוא לא ב-*SLR* היא מכיוון ש-*FOLLOW* של *A* מכיל גם את *b* וגם את *c* ולכן ב-*SLR* יהיה קונפליקט, אבל אם עוקבים אחרי המצבים כפי שעושים ב-*LALR* ו-*LR(1)* הקונפליקט הזה לא קיים, כי *c* או *b* יכולים להופיע אחרי *A* רק אם ראינו או לא ראינו *b* בהתחלה, בהתאמה.

<u>שאלה מספר 20:</u>

נתונות שתי פעולות בינאריות בין מספרים: @,#.

נסמן " # > @ " אם @ בעל קדימות על פני #, ו- " @ > # " אחרת.

נתון הביטוי הבא:

$$0 \# 9 @ 8 @ 7 \# 6 \# 5 @ 4$$

ומתקיים כי סדר הפעולות הנובע מאסוציאטיביות והקדימות של הפעולות הוא:

$$(0 \# 9) @ \Big(8 @ \big(((7 \# 6) \# 5) @ 4\big)\Big)$$

מהו יחס הקדימויות והאסוציאטיביות בין הפעולות?

א. # בעל אסוציאטיביות ימנית, @ בעל אסוציאטיביות שמאלית ו- # > @.

ב. # בעל אסוציאטיביות ימנית, @ בעל אסוציאטיביות שמאלית ו- @ > #.

ג. # בעל אסוציאטיביות שמאלית, @ בעל אסוציאטיביות ימנית ו- @ > #.

ד. # בעל אסוציאטיביות שמאלית, @ בעל אסוציאטיביות ימנית ו- # > @.

<u>פתרון שאלה מספר 20:</u>

מכך ש-# תמיד נמצא בסוגריים הפנימיים ניתן להסיק ש- @ > #.
מהביטוי 5 # (6 # 7) ניתן להסיק ש-# בעל אסוציאטיביות שמאלית.
מהביטוי (4 @ (…)) @ 8 ניתן להסיק ש-@ בעל אסוציאטיביות ימנית.

טור ב'

שאלה	שאלה בטור א'	תשובה נכונה
1	1	ג
2	2	א
3	3	ד
4	4	ב
5	5	ג
6	6	א
7	7	ו
8	8	ה
9	9	ג
10	10	ג
11	11	א
12	13	ג
13	15	ב
14	12	ג
15	14	א+ד
16	16	ד
17	19	ב
18	20	ב
19	17	ב
20	18	א

טור ג'

שאלה	שאלה בטור א'	תשובה נכונה
1	1	א
2	2	ג
3	3	ג
4	5	א
5	6	א
6	6	ג
7	7	ב
8	8	ג
9	9	ב
10	10	ג
11	15	ג
12	13	ד
13	11	ב
14	12	ב
15	14	א+ג
16	16	ב
17	17	ב
18	19	א
19	20	ג
20	18	ב