מבחן סוף סמסטר – מועד ב'

טור 1

מרצה אחראי: ד"ר הילה פלג

מתרגלים: הילה לוי, תומר כהן, אנדריי בבין, תומר ביתן

הוראות:

1. בטופס המבחן 13 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 הטורי הנפרד בלבד ואת התשובות לשאלות הפתוחות יש לכתוב בריבועים הריקים לכל שאלה בטופס התשובות הנפרד.
8. ודאו כי אתם מגישים טופס תשובות וטופס טורי 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 magicVar = 13;
2.  int i = 10;
3.  int k = 2360360;
4.
5.  while (i < 300) {
6.      k = k / i;
7.      i = i + 1b;
8.  }
9.
10. printi(k);
```

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

שאלה מספר 1:

(3 נק') מחליפים את שורה 1 בשורה

```
1. int magic-var = 13;
```

א. שגיאה בייצור קוד

ב. שגיאה בניתוח לקסיקלי

ג. שגיאה בניתוח סמנטי

ד. שגיאה בזמן ריצה

ה. אין שגיאה

ו. שגיאה בניתוח תחבירי

שאלה מספר 2:

(3 נק') מחליפים את שורה 1 בשורה

```
1. int magic_var = 13;
```

א. אין שגיאה

ב. שגיאה בייצור קוד

ג. שגיאה בניתוח תחבירי

ד. שגיאה בזמן ריצה

ה. שגיאה בניתוח סמנטי

ו. שגיאה בניתוח לקסיקלי

שאלה מספר 3:

(3 נק') מחליפים את שורה 2 בשורה

```
2. byte i = 10b;
```

א. שגיאה בניתוח סמנטי

ב. אין שגיאה

ג. שגיאה בזמן ריצה

ד. שגיאה בניתוח לקסיקלי

ה. שגיאה בניתוח תחבירי

ו. שגיאה בייצור קוד

שאלה מספר 4:

(3 נק') מחליפים את שורה 3 בשורה

```
3. byte k = 2360360;
```

א. שגיאה בניתוח תחבירי

ב. שגיאה בניתוח לקסיקלי

ג. שגיאה בייצור קוד

ד. שגיאה בזמן ריצה

ה. שגיאה בניתוח סמנטי

ו. אין שגיאה

שאלה מספר 5:

(3 נק') מחליפים את שורה 7 בשורה

```
7. i++;
```

א. שגיאה בניתוח סמנטי

ב. שגיאה בניתוח תחבירי

ג. אין שגיאה

ד. שגיאה בייצור קוד

ה. שגיאה בניתוח לקסיקלי

ו. שגיאה בזמן ריצה

שינויים ב-FanC:

נרצה להוסיף לשפת FanC את האופרטור = : כך שהביטוי

```
x := y
```

יכניס ל-x את הערך של y אם השמה של x ל-y היא חוקית ואז יחזיר true, ואחרת לא ישנה את ערך המשתנה ויחזיר false.

כך שקטע הקוד הבא:

```
12. if (x:= y+1 or x := y) {
13.   print("success");
14. }
```

- ידפיס success אם x ו-y מוגדרים כ-int, וערכו של x אחרי שורה 14 יהיה אחד יותר מערכו של y.
- ידפיס success אם x ו-y מוגדרים כ-byte, וערכו של x אחרי שורה 14 יהיה כערכו של y.
- לא ידפיס דבר אם x מוגדר byte ו-y מוגדר int, וערכו של x לא ישתנה.

שימו לב כי אם ההשמה הראשונה אינה חוקית, ריצת התוכנית תמשיך לנסות את ההשמה השנייה. שגיאות זמן ריצה בחישוב צד ימין של ההשמה לא משפיעות על ערך ההחזרה אלא נזרקות החוצה מן האופרטור.

שאלה מספר 6:

(6 נק') אילו שלבים של הקומפיילר נצטרך לשנות כדי לתמוך באופרטור החדש?

א. תחבירי, סמנטי

ב. לקסיקלי, תחבירי, סמנטי, ייצור קוד

ג. סמנטי, יצור קוד

ד. לקסיקלי, תחבירי, סמנטי

ה. לקסיקלי, ייצור קוד

ו. לקסיקלי, תחבירי

שאלה מספר 7:

(6 נק') כיצד ניתן למנוע מפעולות קוד ביניים לא תקינות סמנטית להתבצע בזמן חישוב אופרטור = : בזמן ריצה?

א. אין דרך לייצר עבור האופרטור קוד שלא עלול לגרום להתנהגות בלתי-מוגדרת של התכנית

ב. לא ייתכנו פעולות לא תקינות סמנטית בקוד ביניים, שכן אין בו טיפוסים

ג. בדיקות הטיפוסים של שפת הביניים ימנעו מהפעולות להתבצע

ד. ניתן להסתמך על ערך החזרה של הביטוי בזמן ייצור הקוד ולא לייצר פעולות לא תקינות סמנטית

ה. ניתן להסתמך על ערך החזרה של הביטוי בשלב התחבירי ולהחליף את האופרטור בקבוע במחסנית של בייזון

אופטימיזציות (18 נק')

נתונות שורות הקוד הבאות שמהוות בלוק בסיסי בתוך תכנית גדולה יותר:

```
1. y = 1 + 4
2. x = 5
3. c = 1
4. k = y * c
5. x = g + y
6. c = c + 1
7. z = g + k
8. c = z - c
```

ללא כל מידע נוסף, הריצו עליהן עד להתכנסות את האופטימיזציות הבאות:

- constant propagation
- copy propagation
- constant folding
- algebraic simplification
- common subexpression elimination
- useless code elimination

לאחר מכן ענו על השאלות הבאות.

שאלה מספר 8:

(6 נק') אם נתונה לכם **לצורך שאלה זו בלבד** הרצה של אנליזת חיות המוצאת שבסוף הבלוק המשתנים k,y מתים, כמה שורות יישארו בקטע הקוד אחרי ביצוע useless code elimination אחרי האופטימיזציות שכבר בוצעו?

א. 4

ב. 5

ג. 2

ד. 6

ה. 3

שאלה מספר 9:

(6 נק') אחרי הרצת האופטימיזציות ללא מידע נוסף, הפקודה שהייתה במקור בשורה 7 תהיה כעת z = ____ (השלימו את החסר)

א. ניתן למחוק שורה זו לגמרי

ב. x

ג. g + k

ד. 5

ה. g + 5

ו. g + y

שאלה מספר 10:

(6 נק') אחרי הרצת האופטימיזציות ללא מידע נוסף, הפקודה שהייתה במקור בשורה 8 תהיה כעת ____ = c
(השלימו את החסר)

א. 3 - g
ב. 2 - x
ג. c - x
ד. 3
ה. c - z
ו. ניתן למחוק שורה זו לגמרי

חלק ב' - שאלות פתוחות (55 נק')

שאלה 1: דקדוקים (25 נק')

נתון הדקדוק G מעל הטרמינלים s,t והמשתנים E,R, אשר **נמצא** ב-LR(1):

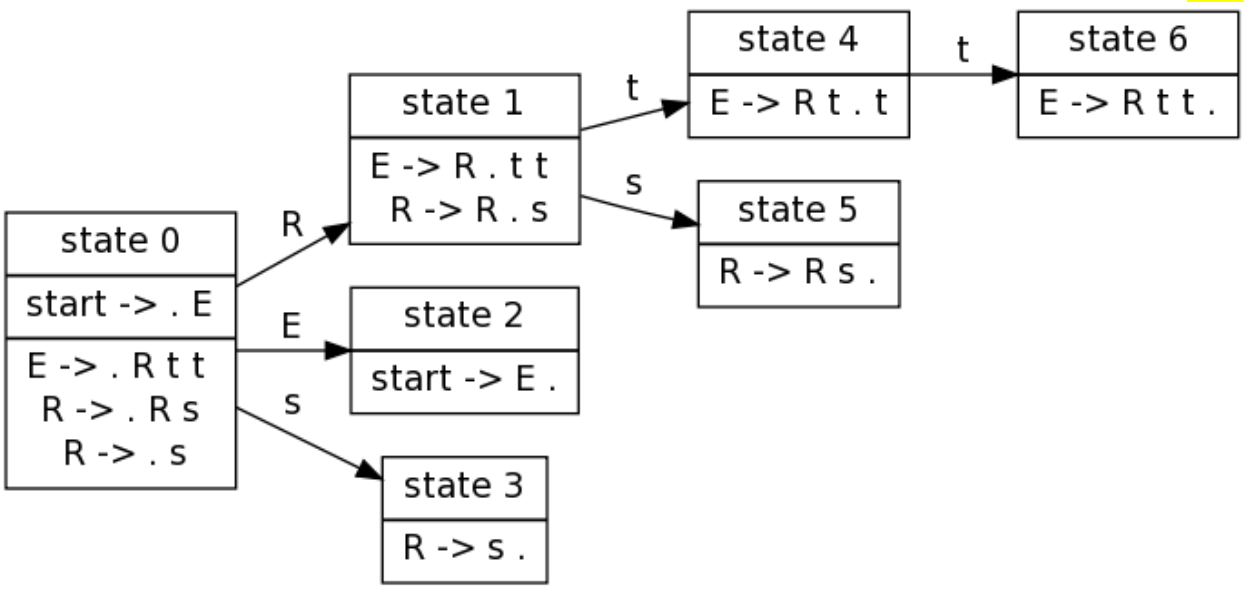
$$E \rightarrow R\ t\ t$$
$$R \rightarrow R\ s \mid s$$

א. (4 נק') מהי השפה המתקבלת מהדקדוק? תנו ביטוי רגולרי אחד בהתאם לכללים שהגדרתם איתם אסימונים ל-flex. הביטוי צריך להתאים לכל המילים בשפה ולהן בדיוק: כל מילה בשפה תתאים לביטוי הרגולרי, ואם מילה מתאימה לביטוי הרגולרי היא בהכרח בשפה.

s+tt

ב. (8 נק') מבין המחלקות LR(0), SLR, LR(1) מהי המחלקה הקטנה ביותר בה נמצא הדקדוק? תזכורת: מחלקה A קטנה יותר ממחלקה B אם מחלקה B יכולה לטפל בכל הדקדוקים בהם יכולה לטפל מחלקה A. נמקו את תשובתכם.

LR(0)



	$	s	t	E	R
0		s3		s2	s1
1		s5	s4		
2	acc				
3	R3	R3	R3		
4			s6		
5	R2	R2	R2		
6	R1	R1	R1		

ג. (2 נק') האם הדקדוק נמצא ב-LL(1)?

לא .

select (R->s) = first(s) = {s}

select (R->Rs) = first(R) = {s}

לפי משפט משום שהחיתוך של הselect של שני חוקים מאותו משתנה לא ריק אז הדקדוק לא בLL1 .

ד. (6 נק') נרצה לוודא שכמות ה-s במילה גדולה מכמות ה-t. כתבו הגדרה מונחית תחביר עבור G המשתמשת בתכונות **נורשות בלבד** שתוודא זאת. הגדירו את התכונות והפעולות הסמנטיות.

הוראות:

- אם המילה תקינה אין לבצע דבר, ואם המילה אינה תקינה יש לקרוא לפונקציה error()
- אין לשנות את הדקדוק! יש לכתוב פעולות סמנטיות רק במקומות המסומנים לכם.
- אין להוסיף לטרמינלים תכונות סמנטיות.

ציינו במפורש איזה תכונות סמנטיות הגדרתם לכל אחד מהמשתנים **ונמקו היטב!!** את משמעות התכונה.

מלאו את מימוש הבדיקה שלכם עבור כל כללי הדקדוק.

יש למלא את ההגדרות במקום המתאים לכך במחברת התשובות.

נשמור לR תכונה שסופרת את מספר הs שיש בביטוי. תמיד יהיה לנו שני t לכן אפשר בסוף לבדוק שהביטוי מכיל לפחות 3 s.

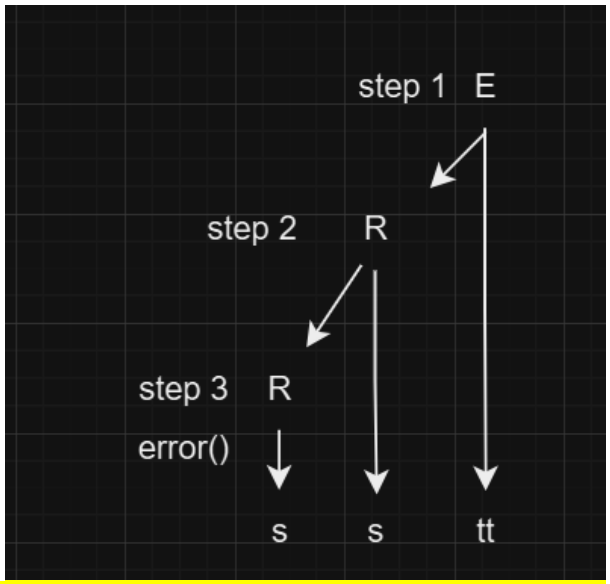
```
E->R1tt {
R1.val = 0
}

R-> R1s {
        R1.val = R.val + 1
}

R-> s {
        if R.val <= 2:
                error()
}
```

ה. (5 נק') נתחו את המילה sstt : הריצו עליה את המנתח של המחלקה הקטנה ביותר שמצאתם בסעיף ב' ואם היא תקינה תחבירית הריצו על עץ הגזירה שלה את הפתרון שלכם לסעיף ד'. מספרו את שלבי הריצה והסבירו בדיוק מתי תימצא השגיאה.

(0,)		
(0,)(1,s)	**s1**	**sstt**
(0,)(2,R)	**r3 + goto 2**	**stt**
(0,)(2,R)(3,s)	**s3**	**stt**
(0,)(2,R)	**R2 + goto 2**	**tt**
(0,)(2,R)(4,t)	**s4**	**tt**
(0,)(2,R)(4,t)(5,t)	**s5**	**t**
(0,)(E)	**acc**	



נריץ את האלגוריתם על העץ אחרי שבנינו אותו. נתחיל מהשורש ונרד לפי כללי הגזירה. לבסוף נגיע לשלב האחרון ונראה שיש לנו מעט מדי s ולכן נזהה שם שגיאה

שאלה 2: אנליזה סטטית (30 נק')

בניסיון לשפר את אלגוריתם הקצאת הרגיסטרים של הקומפיילר שלו, מקס מעוניין להכניס הפרדה בין שני סוגי משתנים: מתים ברובם (mostly dead) או מתים לגמרי (really dead). משתנה מת לגמרי הוא משתנה שאין שימוש בערך שבו באף מסלול בתכנית. משתנה מת ברובו הוא משתנה שקיים לפחות מסלול אחד בתכנית בו אין בו שימוש בערכו יותר.

א. (5 נק') לדעתו של מקס, הוא כבר למד בקורס קומפילציה אנליזה שתחשב משתנים מתים לגמרי. האם מקס צודק? אם כן, הסבירו באיזו אנליזה עליו להשתמש וכיצד עליו לשנות את הפלט שלה. אם לא, הסבירו איזו אנליזה יצטרך לשם כך, והגדירו את הדומיין שלה, יחס הסדר ופעולת ה-join, ופונקציית המעברים.

כן, אנליזת חיות שנלמדה היא בדיוק המשלים של "מת לגמרי", כי היא מחשבת משתנים שעשויים להיות חיים. נריץ את אנליזת חיות ולאחר שתתכנס נחסר את כל הערכים שהתקבלו ב-in/out של הבלוקים מקב' המשתנים Var. לצורך האנליזה נחשיב את print(x) ואת x* כקריאה של x.

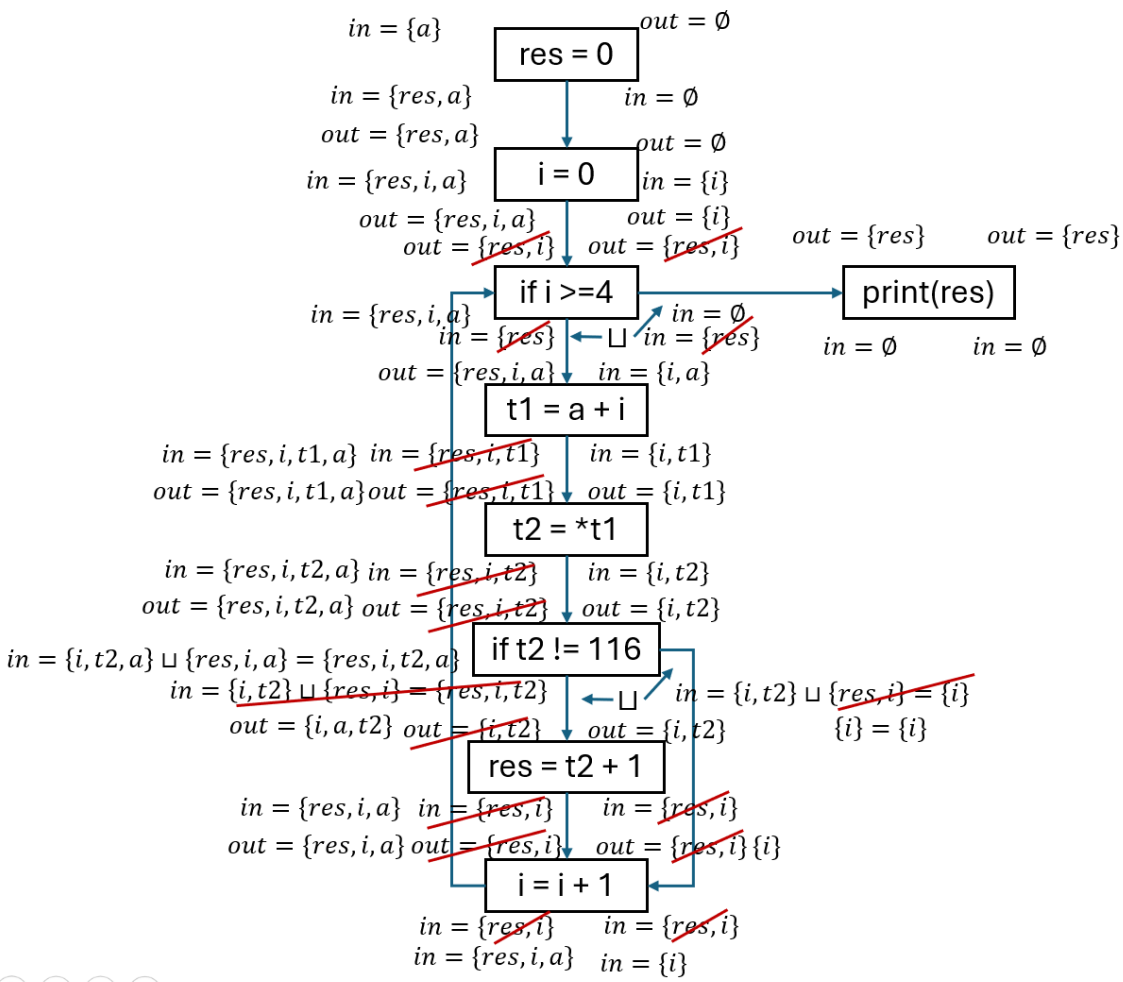
ב. (5 נק') הציעו למקס גם אנליזה שתמצא משתנים מתים ברובם. הגדירו את הדומיין, יחס הסדר, פעולת ה-join, ופונקציית המעברים.

דרושה לנו אנליזה שזהה לאנליזת חיות רק שמבקום להתמודד עם מיזוג מסלולים בתכנית על ידי שמירת כל משתנה שחי בלפחות מסלול אחד, תשמור משתנים שחיים בכל המסלולים. כל הגדרות האנליזה נשארות זהות לחיות, אבל נהפוך את יחס הסדר $\sqsubseteq = \supseteq$ מה שיהפוך את פעולת ה-join שנובעת ממנו: $\sqcup = \cap$. בסוף הרצת האנליזה שוב נחסר את כל הערכים שהתקבלו מ-Var.

ג. (12 נק') נתונה הפונקציה הבאה.
שימו לב: הפעולה x* קוראת מהזיכרון בכתובת השמורה ב-x, כפי שראינו בהרצאה.

```
1.  res = 0
2.  i = 0
loop:
3.  if i >= 4 goto loop_end
4.  t1 = a + i
5.  t2 = *t1
6.  if t2 != 116 goto loop_post
7.  res = t2 + 1
8.  goto loop_post
loop_post:
9.  i = i + 1
10. goto loop
loop_end:
11. print(res)
```

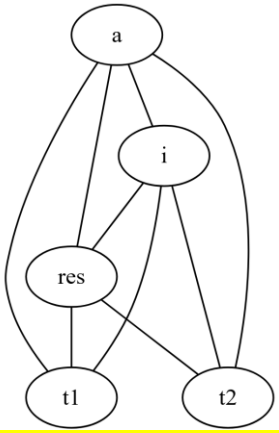
ציירו את ה-CFG של הפונקציה והריצו את האנליזות שהגדרתם בסעיפים א' וב'. אם האנליזות שלכם דורשות חישוב נוסף על התוצאה כדי לקבל את המשתנים המתים ברובם והמתים לגמרי, כדאי לבצע אותו בצד אחרי התכנסות האנליזה.



סעיף א' (חיות רגיל) מצד שמאל, סעיף ב' (חיות עם חיתוך) בצד ימין.
במצב זה אחרי התכנסות האנליזה ניתן לחסר את כל הערכים הסופיים מ-Var.

ד. (5 נק') ציירו את גרף ה-interference של הפונקציה לפי תוצאת האנליזה שלכם. להזכירכם: הצמתים הם משתנים בתכנית ואם שני משתנים לא יכולים להשתמש באותו הרגיסטר באותו הזמן מכיוון שהערכים של שניהם עדיין נחוצים בנקודה כלשהי בתכנית, תהיה ביניהם קשת בגרף.

אם שני ערכים נחוצים באותו זמן בתכנית, אז הם live באותו הזמן. לכן נצייר צומת לכל משתנה וקשת בין כל שני משתנים שנמצאים ביחד בתוצאה של אנליזת חיות באיזשהו בלוק.



כאשר בעצם הקשת החסרה היחידה היא בין t1 ל-t2

ה. (3 נק') מקס טוען שבעת הקצאת רגיסטרים בצביעה, בחירת מועמד ל-spill **שאינו** מת ברובו או מת לגמרי יהיה עדיף על מועמד ל-spill שהוא מת ברובו. האם מקס צודק? הסבירו **בקצרה** את ההיגיון של תשובתכם על המשתנה res בשורה 3.

כשאנחנו בוחרים מועמד ל-spill אנחנו מנסים למצוא משתנה שהוספה של שמירה לזיכרון וקריאה מהזיכרון עבורו תשפיע הכי לטובה שאפשר על גרף ה-interference, כלומר תייצר גרף שאפשר לצבוע אותו בפחות צבעים. אם משתנה מת לגמרי בנקודה כלשהי בתכנית, אז הנקודה הזו לא מייצרת קשת שתפריע לנו. אם הוא מת ברובו, ונכניס כתיבה שלו לזיכרון, יש מסלולים בתכנית שבהם הוא לא מייצר לנו קשתות לגרף. התיאוריה של מקס היא שניקח משתנה שלא מת ברובו, כלומר משתמשים בו בכל מסלול בתכנית, ובכך נייצר השפעה הכי גדולה על הגרף. מצד שני, אם נסתכל על res בשורה 3, הוא רק מת ברובו, אבל לשפוך אותו יגרום לשינוי עצום בגרף. אז, בעוד שניתנו נקודות לכל עמדה מנומקת היטב ביחס לתוצאות, אפשר לראות שהיוריסטיקה של מקס לא משהו, ואם הוא יישם אותה ויצפה שהקומפיילר ייצר קוד יותר יעיל, יידרש נס.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<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:

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

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:

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

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:

אסימון	תבנית
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	+ | - | * | /
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
STRING	"([^\n\r\"\\]|\\[rnt"\\])+"

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. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$