27.2.2018

מבחן סוף סמסטר – מועד ב'

מרצה אחראי: ד"ר שחר יצחקי

מתרגלים: אבנר אליזרוב, עומר כץ, הילה פלג

הוראות:

א. בטופס המבחן 12 עמודים מהם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן שלוש שעות (180 דקות).

ג. במבחן 6 שאלות. כל השאלות הינן חובה. משקל כל שאלה מופיע בראשיתה. (חלוקת המשקל בין הסעיפים בכל שאלה אינה בהכרח אחידה.)

ד. ניתן לציין לגבי סעיף או שאלה "לא יודע/ת". תשובה זו תזכה ב- 20% מהניקוד של הסעיף או השאלה. תשובות שגויות לא יזכו בניקוד.

ה. חובה לנמק כל תשובה. לא יינתן ניקוד על תשובות ללא נימוק.

ו. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.

ז. אין צורך להגיש את הטופס בתום הבחינה.

ח. את התשובות לשאלות יש לרשום במחברת הבחינה בלבד.

בהצלחה!

שאלה 1 (20 נקודות): שלבי הקומפילציה

חלק א - סיווג מאורעות (10 נקודות)

נתון קטע הקוד הבא בשפת FanC:

```
1.   int func1(int a, int c){
2.       return a / c;
3.   }
4.   int func2(){
5.       int a = 300;
6.       int c = 4;
7.       return func1(a, a - c) + a;
8.   }
9.   void main(){
10.      //comment
11.      int res = func2();
12.      return;
13.  }
```

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) ל-main של התוכנית. עבור כל שינוי **כתבו** האם הוא גורם לשגיאה. אם כן, **ציינו** את השלב המוקדם ביותר שבה נגלה אותה (ניתוח לקסיקלי, ניתוח תחבירי, ניתוח סמנטי, ייצור קוד, זמן ריצה) **ונמקו בקצרה**:

1. מחליפים את שורה 5 בשורה הבאה:

```
5. byte a = 300b;
```

2. מחליפים את שורה 5 בפקודה הבאה:

```
5. int* a = 4;
```

3. מחליפים את שורה 7 בפקודה הבאה:

```
7. return func1(a) + &a;
```

4. מחליפים את שורה 11 בפקודה הבאה:

```
11. bool res = func2();
```

5. מוחקים את שורה 12.

חלק ב – הרחבת השפה (10 נקודות)

הנכם מתבקשים להוסיף **לשפת FanC** יכולת חדשה. קראו את תיאור היכולת, ופרטו בקצרה איזה שינוי צריך להתבצע בכל שלב בקומפילציית השפה. **התייחסו לשלבים לקסיקלי, תחבירי, סמנטי, ייצור קוד אסמבלי (שפת ביניים)**. הקפידו על ההפרדה בין השלבים.

נרצה להוסיף לשפת FanC תמיכה בביטויים טרנריים.

ביטויים טרנריים הם ביטויים מהצורה $\mathbf{x}\ ?\ \mathbf{y}\ :\ \mathbf{z}$, כך ש$\mathbf{x}$ הוא תנאי ו$\mathbf{y}, \mathbf{z}$ הם ביטויים המחזירים ערך.
ערך הביטוי הטרנרי יהיה ערכו של $\boldsymbol{y}$ אם התנאי $\boldsymbol{x}$ מתקיים. אחרת ערכו יהיה כערך $\boldsymbol{z}$.
כלומר, בהשמה $\boldsymbol{int\ a\ =\ x\ ?\ y\ :\ z}$ תוצאת ההשמה ל-$\boldsymbol{a}$ תהיה שקולה להפעלת המשפט $\boldsymbol{if(x)\ \{\ a\ =\ y;\ \}\ else\ \{\ a\ =\ z;\ \}}$

לדוגמה:

$$(\boldsymbol{a} > \mathbf{1})\ ?\ \mathbf{1}\ :\ \mathbf{2}$$

ישוערך ל$\mathbf{1}$ אם מתקיים $\boldsymbol{a} > \mathbf{1}$, אחרת ישוערך ל$\mathbf{2}$.

ניתן לקנן ביטויים טרנריים. למשל, הביטוי $\mathbf{x}\ ?\ \mathbf{y}\ :\ \mathbf{a}\ ?\ \mathbf{b}\ :\ \mathbf{c}$ הוא ביטוי חוקי והמשפט $\mathbf{return\ x\ ?\ y\ :\ a\ ?\ b\ :\ c}$ יפורש כך:

```
if(x)
  return y;
else
  if (a)
    return b;
  else
    return c;
```

ביטויים מורכבים כדוגמת $\mathbf{x}\ ?\ \mathbf{y}\ :\ \mathbf{z}\ +\ \mathbf{w}$ יפורשו כ $(\mathbf{x}\ ?\ \mathbf{y}\ :\ \mathbf{z})\ +\ \mathbf{w}$.

שימו לב:

- $\mathbf{y}, \mathbf{z}$ הם ביטויים בעלי ערך ולא משפטים
- $\mathbf{x}$ הוא תנאי בוליאני
- ביטוי טרנרי יכול להיות מטיפוס מספרי או בוליאני. אם הביטוי הוא מספרי, טיפוסו יקבע בהתאם לטיפוסי $\mathbf{y}$ ו$\mathbf{z}$ (בדומה לביטויים אריתמטיים).
- רק אחד מבין $\mathbf{y}, \mathbf{z}$ ישוערך.

שאלה 2 (30 נקודות): אנליזה סטטית

השאלה מתבססת על שפת WHILE, הנתונה על-ידי הדקדוק הבא:

$$
\begin{aligned}
S &\rightarrow \text{id} := E \mid S\,;\,S \mid \text{skip} \\
&\mid \text{if } E \text{ then } S \text{ else } S \\
&\mid \text{while } E \text{ do } S \\
E &\rightarrow \text{id} \mid \text{num} \mid E \;\square\; E \qquad (\square \in \{+, -, *, /, =, \neq, <, >, \leq, \geq\})
\end{aligned}
$$

מרחיבים את השפה כך שתכלול גם מערכים (חד-ממדיים). לשם כך מוסיפים כללי גזירה:

$$
\begin{aligned}
S &\rightarrow \text{id}[E] := E \\
E &\rightarrow \text{id}[E] \mid \text{new}\,[E]
\end{aligned}
$$

מערכים מוקצים באופן דינמי, וכידוע הקצאות דינמיות ללא שחרור מפורש מחייבות שימוש בשיטות של garbage collection למניעת דליפת זכרון. הדבר מתבטא בתקורת זמן ריצה.

חוקרים מאוניברסיטת MIT חקרו ומצאו כי מרבית המערכים נמצאים בשימוש בקטעי קוד קצרים בלבד. הם מציעים את הייעול הבא: בכל scope התחום בסוגריים עגולים המתכנת יצהיר על משתנים מקומיים, כך –

$$S \rightarrow \text{local id}$$

(local תהיה מעתה מילה שמורה בשפה.) הקומפיילר יוודא – בשלב הבדיקות הסמנטיות – שמשתנים אלה אינם בשימוש מחוץ לגבולות ה-scope הנתון. בסוף כל scope, הקומפיילר יוסיף קוד לשחרור המשתנים המקומיים.

לדוגמה, בתכנית שלהלן:

```
q := new [n];
i := 0;
while (i < n) do (
      local p;
      p := new [3];
      p[0] := i * i;
      p[1] := p[0] * p[0];
      p[2] := p[1] * p[1];
      q[i] := p[0] + p[1] + p[2]
)
```

המערך המוצבע על-ידי p ישוחרר בסוף כל איטרציה של הלולאה, שכן p מוגדר כמשתנה מקומי בגוף הלולאה; ואילו המערך המוצבע על-ידי q לא ישוחרר וייאסף, כרגיל, על-ידי garbage collection.

הערות לכלל הסעיפים:

- ניתן להשתמש בכל אנליזה סטטית שנלמדה בכיתה בציון שמה ודרך השימוש בה.
- ניתן להגדיר שורה בודדת כבלוק בסיסי (יש לציין זאת).
- יש לציין מהו ה-abstract domain, ומהן פונקציות האבסטרקציה והקונקרטיזציה. יש להגדיר את הסמנטיקה האבסטרקטית המתאימה. ניתן להפנות להגדרות מחומר הקורס <u>ואין צורך</u> להגדירן מחדש.

א. (5 נק') בהצעה זו קיים **פגם מהותי**. מהו? הדגימו באמצעות תכנית מתאימה.

ב. (15 נק') הציעו שיפור, המבוסס על אנליזה סטטית, לתיקון הפגם שזיהיתם.

ניתן להניח קיום הבדיקות הסמנטיות שבגרסה הפגומה, וכן טבלת סמלים המכילה את שמות כל המשתנים המקומיים (אלה המוצהרים כ-local) עבור כל scope.

ג. (10 נק') החוקרים מ-MIT טוענים שהאנליזה שלכם **אינה מדויקת לחלוטין** ("loss of *precision*", "the analysis is not *complete*"). הם צודקים, כמובן (איך ידענו?). הסבירו את משמעות אי-הדיוק ושוב, הדגימו בעזרת תכנית.

<u>שאלה 3 (10 נקודות): אופטימיזציה</u>

הסעיפים הבאים מתייחסים למנגנון הJIT (Just-In-Time Compilation) שנלמד בתרגולים:

א. **(5 נקודות)** סטודנט טען כי ניתן לייעל את מנגנון הJIT ע"י הרצה של התכנית פעם אחת ושמירת הקוד המיוצר מהJIT. בהרצות הבאות נשתמש בקוד שייצר הJIT מההרצה הראשונה ולא נבצע JIT מחדש. בכך נחסוך את התקורה שלו לזמן הריצה. האם הייעול הנ"ל ישמור על ביצועים דומים למנגנון המקורי תמיד? אם כן, הסבירו מדוע. אם לא, תנו דוגמא לתכנית שבה זה לא יעבוד.

(5 נקודות) בעת ביצוע אופטימיזציות JIT מתקדמות, מנגנון ה-JIT מתבקש לטפל בפונקציה קצרה שמקבלת מספר רב של פרמטרים ונקראת מספר גדול מאוד של פעמים בזמן ריצה. מן המידע שנאסף בזמן ריצה, ידוע כי זמן בניית ה-Activation Record של הפונקציה ארוך יותר מזמן ביצועה, וכי הפונקציה רק קוראת ולא כותבת לתוך אף אחד מהפרמטרים שלה. הציעו אופטימיזציה לתכנית (לאו דווקא רק לפונקציה עצמה) שתייעל את ביצוע הפונקציה.

<u>שאלה 4 (15 נקודות): ניתוח תחבירי וסמנטי</u>

א. (5 נק') נתון הדקדוק הבא:

$$S \rightarrow a\ b\ D\ E \mid a\ b\ E\ F$$
$$D \rightarrow x$$
$$E \rightarrow y$$
$$F \rightarrow z$$

כאשר אותיות קטנות מייצגות טרמינלים ואותיות גדולות מייצגות משתנים. הדקדוק ניתן לניתוח על ידי מנתח LR(0). המתכנת טוני בנה מנתח LR(0) על ידי קומפיילר מנתחי LR(0) המתייחס לכללים סמנטיים באופן זהה לבייסון. טוני מנסה לדבג את תהליך הניתוח ולכן מוסף את ההדפסות הבאות:

$$D \rightarrow \{printf(\text{"D"});\}\ x$$
$$E \rightarrow \{printf(\text{"E"});\}\ y$$

הסבירו מדוע טוני מקבל שגיאה בבניית המנתח מחדש אחרי הוספת הכללים. הראו את השגיאה.

ב. (2 נק') ברוס מנסה לתקן את השגיאה שטוני קיבל על ידי החלפת ההודעות:

$D \rightarrow \{printf(\text{"X"});\}\ x$
$E \rightarrow \{printf(\text{"X"});\}\ y$

האם השגיאה תוקנה? נמקו.

ג. (8 נק') בעת פיתוח הקומפיילר המסחרי הראשון לשפת FanC, הסטרטאפיסטים הצעירים סטיב ובאקי עמלו על הבדיקות הסמנטיות של השפה. סטיב כתב את הכלל הסמנטי הבא:

$Statement \rightarrow Type\ ID\ ASSIGN\ Expr\ SC$ {
```
      Statement.usedVariables = new Dictionary();
      Statement.usedVariables.add(ID,Type.type);
      checkTypeMismatch(Statement.usedVariables,Expr.usedVariables);
      Statement.usedVariables.addAll(Expr.usedVariables);
```
}

המעביר כלפי מעלה בעץ את הטיפוס של המשתנה שהוצהר וטען כי אם כל ביטוי וכל משפט בשפה יעביר כלפי מעלה בעץ את המשתנים שבשימוש בו וטיפוסיהם, אפשר לגלות TypeMismatch בעת ביצוע איחוד בין שני בנים של צומת בעץ אם תתגלה סתירה, כך:

```
void checkTypeMismatch(Dictionary lhs, Dictionary rhs) {
      for(string id in lhs.keys) {
            if (rhs.contains(id)){
                  type type1 = lhs.getValue(id);
                  type type2 = rhs.getValue(id);
                  if (type1!=type2) typeMismatchError();
            }
      }
}
```

סטיב מצהיר כי בשיטה זו ניתן יהיה לבצע את בדיקות הטיפוסים **ללא שימוש באף משתנה גלובאלי** אלא בעזרת העברת מידע **כלפי מעלה בעץ בלבד**. באקי טוען שסטיב טועה, ושבגישה שלו יש מספר רב של כשלים.

1. האם ניתן יהיה לקבוע את הטיפוס של ביטויים בשיטה של סטיב? אם כן, הסבירו כיצד. אם לא, תנו דוגמה שתיכשל והסבירו היכן תהיה הבעיה.
2. האם ניתן לבדוק תקינות של השמה ושל טיפוס ערך ההחזרה של פונקציה בשיטה של סטיב? אם כן, הסבירו כיצד. אם לא, תנו דוגמה שתיכשל והסבירו היכן תהיה הבעיה.
3. האם ניתן לבדוק כי לא משתמשים במשתנה לפני שהוגדר בשיטה של סטיב? אם כן, הסבירו כיצד. אם לא, תנו דוגמה שתיכשל והסבירו היכן תהיה הבעיה.

<u>שאלה 5 (10 נקודות): רשומות הפעלה</u>

א. (3 נק') מעבדים מודרניים מחלקים את הרגיסטרים לקבוצה שבאחריות הצד הקורא לגבות (caller-save) וקבוצה שבאחריות הצד הנקרא לגבות (callee-save). הסבירו מה החסרון בכך שכל הרגיסטרים יהיו באחריות אותו צד?

ב. (4 נק') האם ניתן לממש רשומות הפעלה כך שתמיד כתובת החזרה תידחף למחסנית לפני הארגומנטים המועברים לפונקציה הנקראת? הניחו שבמעבד הרלוונטי נדרש לשמור את כתובת החזרה באופן ידני.

ג. (3 נק') בתרגול למדנו שבשלב ייצור הקוד משתמשים ברגיסטר sp כדי לפנות לראש המחסנית וברגיסטר fp כדי לגשת למשתנים וארגומנטים. האם ניתן לממש ייצור קוד לשפת FanC מבלי להשתמש בfp ומבלי לשנות את הדקדוק ו/או הסמנטיקה של השפה?

שאלה 6 (15 נקודות): Backpatching

נתון כלל הדקדוק עבור הביטוי naughtOneInfty:

$E \rightarrow \text{naughtOneInfty}(BL)\ \{E_1, E_2, E_3\}$
$BL \rightarrow BL, B$
$BL \rightarrow B$

הביטוי naughtOneInfty יחושב לפי הסמנטיקה הבאה:
רשימת הביטויים הבוליאניים BL תשוערך משמאל לימין כל עוד כל הביטויים ששוערכו הם אמת. כאשר הסתיימה הרשימה או ביטוי כלשהו קיבל ערך שקר, ייבחר ערך הביטוי מתוך הרשימה ב-EL לפי מספר הביטויים שערכם היה אמת: אם אפס ביטויים בוליאניים שוערכו לפני שביטוי קיבל ערך שקר, ערך הביטוי כולו יהיה כערך E_1. אם ביטוי אחד שוערך לפני שביטוי קיבל ערך שקר, ערך הביטוי כולו יהיה כערך E_2.
עבור כל מספר אחר של ערכי אמת, ערך הביטוי יהיה ערך E_3.
ביטויי E שאינם הביטוי שערכו נבחר לא יחושבו, ולא יחושבו עוד ביטויים בוליאניים אחרי הביטוי הראשון שערכו שקר.

לדוגמא, עבור הקוד:

```
int a = naughtOneInfty(true, false, 1+2==3,true,3>4){
    10+3,
    5,
    x
};
```

ערך המשתנה a יהיה 5.

א. (5 נק') הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות שאתם משתמשים לכל משתנה.

ב. (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.

שימו לב:

- אין לשנות את הדקדוק
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- ניתן להשתמש במרקרים N,M שנלמדו בכיתה בלבד.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנים S,B,E ישנן התכונות שהוגדרו בכיתה בלבד. לאסימון NUM אין תכונות פרט למוגדר בשאלה.
- למשתנים S,B,E יש כללי גזירה פרט לאלו המוצגים בשאלה.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

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 SHIFT
                else ERROR
        else    // X ∈ V
                if M[X , t] = error then ERROR
                else REPLACE(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
```

ייצור קוד בשיטת Backpatching

פונקציות:

`makelist(quad)`	יוצרת רשימה ריקה עם איבר אחד (ה"חור" quad).
`merge(list1,list2)`	מחזירה רשימה ממוזגת של הרשימות list1, list2
`emit(code string)`	מדפיסה קוד בשפת הביניים ומאפשרת להדפיס פקודות קפיצה עם "חורים".
`nextquad()`	מחזירה את כתובת הרביעיה (הפקודה) הבאה שתצא לפלט.
`backpatch(list, quad)`	מקבלת רשימת "חורים" list וכתובת quad, ו"מטליאה" את הרשימה כך שבכל החורים תופיע הכתובת quad.
`newtemp()`	מחזירה שם של משתנה זמני חדש שאינו נמצא בשימוש בתכנית.

משתנים סטנדרטיים:

- S: גוזר פקודות (statements) בשפה. תכונות:
  - nextlist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת הפקודה הבאה לביצוע אחרי הפקודה הנגזרת מ- S.
- B: גוזר ביטויים בוליאניים. תכונות:
  - truelist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני מתקיים.
  - falselist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני אינו מתקיים.
- E: גוזר ביטויים אריתמטיים. תכונות:
  - E.place: שם המשתנה הזמני לתוכו מחושב הביטוי האריתמטי.

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`
7. פקודה ריקה	`nop`

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:

$$out(B) = \bigcap_{(B,S)\in E} in(S) \quad \text{או} \quad out(B) = \bigcup_{(B,S)\in E} in(S)$$

$$in(B) = f_B\big(out(B)\big)$$

Analysis and Optimization

Dominator:
In control flow graph, node *D* **dominates** a node *N* if every path from the entry node to *N* must go through *D*.
Post-dominator:
In control flow graph, node *Z* **post dominates** a node *N* if all paths to the exit node of the graph starting at *N* must go through *Z*.

דקדוק שפת FanC

$Program \rightarrow Funcs$
$Funcs \rightarrow \epsilon$
$Funcs \rightarrow FuncDecl\ Funcs$
$FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
$RetType \rightarrow Type$
$RetType \rightarrow VOID$
$Formals \rightarrow \epsilon$
$Formals \rightarrow FormalsList$
$FormalsList \rightarrow FormalDecl$
$FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
$FormalDecl \rightarrow Type\ ID$
$Statements \rightarrow Statement$
$Statements \rightarrow Statements\ Statement$
$Statement \rightarrow LBRACE\ Statements\ RBRACE$
$Statement \rightarrow Type\ ID\ SC$

$Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow Call\ SC$
$Statement \rightarrow RETURN\ SC$
$Statement \rightarrow RETURN\ Exp\ SC$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
$Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow BREAK\ SC$
$Statement \rightarrow SWITCH\ LPAREN\ Exp\ RPAREN\ LBRACE\ CaseList\ RBRACE\ SC$
$CaseList \rightarrow CaseList\ CaseStatements$
$CaseList \rightarrow CaseStatements$
$CaseStatements \rightarrow CaseDec\ Statements$
$CaseStatements \rightarrow CaseDec$
$CaseDec \rightarrow CASE\ NUM\ COLON$
$CaseDec \rightarrow CASE\ NUM\ B\ COLON$
$CaseDec \rightarrow DEFAULT\ COLON$
$Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
$Call \rightarrow ID\ LPAREN\ RPAREN$
$ExpList \rightarrow Exp$
$ExpList \rightarrow Exp\ COMMA\ ExpList$
$Type \rightarrow INT$
$Type \rightarrow BYTE$
$Type \rightarrow BOOL$
$Exp \rightarrow LPAREN\ Exp\ RPAREN$
$Exp \rightarrow Exp\ BINOP\ Exp$
$Exp \rightarrow ID$
$Exp \rightarrow Call$
$Exp \rightarrow NUM$
$Exp \rightarrow NUM\ B$
$Exp \rightarrow STRING$
$Exp \rightarrow TRUE$
$Exp \rightarrow FALSE$
$Exp \rightarrow NOT\ Exp$
$Exp \rightarrow Exp\ AND\ Exp$
$Exp \rightarrow Exp\ OR\ Exp$
$Exp \rightarrow Exp\ RELOP\ Exp$