26.2.2016

מבחן סוף סמסטר – מועד ב'

מרצה אחראי: פרופ' ארז פטרנק

מתרגלים: יורי משמן עדי סוסנוביץ עידן שורץ

הוראות:

א. בטופס המבחן 10 עמודים מהם 4 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן שלוש שעות (180 דקות).

ג. אסור כל חומר עזר פרט לדף הנוסחאות המצורף לבחינה.

ד. במבחן 5 שאלות. כל השאלות הינן חובה. משקל כל שאלה מופיע בראשיתה. (חלוקת המשקל בין הסעיפים בכל שאלה אינה בהכרח אחידה.)

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

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

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

ח. אין צורך להגיש את הטופס בתום הבחינה.

ט. את התשובות לשאלות יש לרשום במחברת הבחינה בלבד.

בהצלחה!

שאלה 1 (14 נקודות): סיווג מאורעות

האם המאורעות הבאים קורים בזמן קומפילציה, בזמן ריצה או בזמן בניית הקומפיילר? אם מדובר על זמן קומפילציה כתבו באיזה שלב של הקומפילציה המאורע קורה. נמקו בקצרה.

1. (2 נק') מתבצע spilling של רגיסטר לזיכרון כדי להעלות עליו ערך של משתנה אחר.
2. (2 נק') מתגלה שהמתכנת כתב if מקונן בתוך if עם else יחיד, והמנתח מחזיר שגיאה כי לא ניתן לדעת אם לשייך את הelse ל if החיצוני או הפנימי.
3. (2 נק') מתגלה שהמתכנת נתן לפונקציה שם המתחיל ב"%" ו % לא תו מוכר בשפה.
4. (2 נק') מתגלה שהמשתמש שכח לסגור סוגריים מסולסלים בענף if.
5. (2 נק') מתגלה שהמשתמש שם גרשיים מקוננות למרות שבשפה זה אסור.
   (כלומר מופיע הביטוי: " " " ")
6. (2 נק') מתגלה שגיאה בקריאה למתודה x.foo() כי המשתנה x מטיפוס int בשפה dynamically typed.
7. (2 נק') מתגלה שמספיק 10 רגיסטרים בשביל כל הערכים החיים בנקודה מסויימת בתוכנית.

שאלה 2 (23 נקודות):

בשאלה זו נבנה אלגוריתם Copying לניהול זיכרון דינאמי.

א) (3 נק') תארו יתרון אחד וחיסרון אחד של אלגוריתם ה-copying .

בכיתה תואר ההליך הבא שהוא המרכזי בתהליך האיסוף. מתחילים מהשורשים ומבצעים tracing של האובייקטים החיים (בדומה לשלב הmark באלגוריתם mark and seep), ומעתיקים אוביקטים נגישים אל מחצית ה-heap שנשמרה עבור כך. שימו לב שלאחר העתקת האובייקטים יש לעדכן גם את המצביעים על מנת שיצביעו למקום החדש של האובייקט המועתק. השטח שממנו מעתיקים נקרא from-space והשטח השמור שאליו מעתיקים נקרא to-space.

ב) (5 נק) נהוג להשאיר בכל אובייקט שהועתק מ from-space מחוון אל העותק של האובייקט הזה ב to-space). המחוון הזה נקרא forwarding pointer.
תארו דרך לעדכן את המחוונים בכל אובייקטים שהועתקו תוך שימוש ב-forwarding pointers.
ההסבר שלכם צריך לכלול את הנקודה באלגוריתם שבה מכניסים את ה- forwarding pointes לתוך האובייקט המקורי ואחר כך הדגישו מדוע הוא מועיל בתהליך עדכון המחוונים של האובייקטים המועתקים.

שימו לב שבנוסף לעלות המקום ב-heap (שחציו מבוזבז) יש גם עלות מקום במחסנית הדרושה למעבר מסוג DFS על האובייקטים הנגישים. בהמשך נרצה לחסוך את המקום הדרוש עבור המחסנית.

ג) (3 נק) נניח שהעתקנו את כל האובייקטים הנגישים ישירות מהשורשים אל תחילת to-space וגם עדכנו את השורשים להצביע למקומות החדשים של האובייקטים. האם נכון בשלב זה לומר שכל האובייקטים החיים ב from-space נגישים מן האובייקטים שהועתקו לתחילת to-space?
(הכוונה ב"נגיש" הוא דרך מסלול מחוונים שעובר דרך אובייקטים ב from-space). אם כן- נמקו. אם לא הביאו דוגמה נגדית.

ד) (4 נק) כיוון שהעתקנו אובייקטים מ from-space בלי לשנות בהם כלום, כל המחוונים שלהם כרגע מופנים לאובייקטים ב from-space. נרצה לתקן אותם כך שיצביעו לאובייקטים המתאימים ב to space. נתחיל במעבר על האובייקט הראשון ונתקן את המחוונים שלו אחד אחד כך שיצביעו ל to-space בצורה הבאה:

a. (1 נק) אם האובייקט המוצבע כבר הועתק לto-space פשוט נתקן את המחוון. הסבירו איך יודעים לאן האובייקט עבר.
b. (3 נק) אם האובייקט טרם הועתק, נעתיק אותו למקום הפנוי הבא בto-space, נשתול forwarding pointer, ונתקן את המחוון.
האם ניתן לומר שכל האובייקטים החיים ב from-space נגישים מן האובייקטים שהועתקו עד עתה ל to-space. אם כן- נמקו. אם לא תארו דוגמה נגדית.

ה) (5 נק) שימו לב שלא השתמשנו במקום נוסף מחוץ לheap כדי לתקן את האובייקט הראשון (פרט אולי לשני מחוונים שאומרים איפה המקום הפנוי הבא ב-to-space ואיזה שדה אנחנו מתקנים כרגע).
האם ניתן להמשיך בהעתקות ועידכונים כך ולהעתיק את כל האובייקטים הנגישים ב -from space ? תארו בקצרה איך תהליך כזה מתבצע. האם תהליך כזה מעתיק את כל האובייקטים הנגישים? אם כן - נמקו. אם לא,– הסבירו על ידי דוגמא בעייתית.
ו) (3 נק) כיצד יודעים שהתהליך מהסעיף הקודם הסתיים? האם מובטח שהאלגוריתם בסעיף הקודם תמיד יסתיים? נמקו.

<u>שאלה 3 (23 נקודות): שאלת DFA (אנליזה סטטית)</u>

בשפה חדשה יש טיפוסים דינמיים (כלומר, הטיפוס של משתנה יכול להשתנות בזמן הריצה) והקומפיילר מנסה להסיק את הטיפוסים לבדו (כשהוא יכול). נתבונן בתוכנית שמשתמשת בשלוש מחלקות: A, B, C. לכל אחת מהמחלקות יש שלוש מתודות חסרות פרמטרים:
למחלקה A יש 3 מתודות fun1(), fun2(), fun3() ;
למחלקה B יש 3 מתודות fun5(), fun4(), fun3() ;
ולמחלקה C יש 3 מתודות fun6(), fun2(), fun4().

בהינתן שבשפה כל טיפוסי המשתנים הדינאמיים מוגבלים לאחד הטיפוסים A, B, ו-C, <u>רוצים לתפוס שגיאות</u> מהסוג:
בעת קריאה של משתנה למתודה (לדוגמה x.fun1()) הטיפוס הדינמי של המשתנה לא מתאים (כלומר למשתנה אין את המתודה הזו).
לדוגמה: היו קריאות x.fun1() ו x.fun5() שיבוצעו ברצף בזמן ריצה. אין שום טיפוס שיש לו את שתי המתודות האלה, ולכן לא ברור אם x מטיפוס A או מטיפוס B אבל בכל מקרה תהיה שגיאה באחת משתי הקריאות – עלינו לדווח שגיאה בהגעה לקריאה המופיעה שניה ברצף (ב- x.fun5()).

ערכי החזרה של כל המתודות הן או ערך בוליאני או אחת המחלקות A, B, C, אבל טיפוס הערך המוחזר לא ידוע בזמן הקומפילציה, כלומר ערכי החזרה ידועים רק בזמן ריצה.

קוד לדוגמה:

```
Fun foo (x, y, m, n){

        1: if (  x.fun1()  ) goto 5

        2: z = y.fun2()

        3: y = z.fun6()

        4: goto 7

        5: z = x

        6: y = n.fun3()

        7: x = m.fun6()

        8: if (  z.fun1() ) goto 1

}
```

השפה הנתונה כוללת פקודות של השמות למשתנים של משתנים אחרים או ערכי חזרה של מתודות, ופקודות של קפיצות מותנות או לא מותנות. תנאים של הקפיצות המותנות יכילו רק קריאות למתודות.

1. (6 נק׳) הקדמה

א. (3 נק׳) בצעו חלוקה לבלוקים בסיסיים של קוד הפונקציה בדוגמה, תנו שם לכל בלוק לפי מספר השורה (בקוד המקורי) של הפקודה הראשונה בבלוק, וציירו CFG.

ב. (3 נק׳) ציינו ליד כל פקודה ב-CFG, בה זה רלוונטי, מה הטיפוסים האפשריים של כל משתנה בהתאם לקריאות למתודות שייתכן והתבצעו לפני הפקודה, או בפקודה הזו, וציינו אם קריאה למתודה מסויימת אינה חוקית.

2. (9 נק׳) הראו אנליזת DFA שמהתוצאות שלה ניתן לדווח מתי קריאה מסוימת למתודה יכולה להיות לא חוקית, והסבירו בקצרה כיצד ניתן לזהות שגיאה בשורה מסוימת בעזרת אנליזה זו.

<u>הערות</u>

- על האנליזה להיות עבור CFG בו כל בלוק הוא בלוק בסיסי
- בסעיף זה אל תנסו להסיק את ערכי החזרה של מתודות. והניחו כי ההתייחסות לערכי החזרה של מתודות חוקית בכל מקום (בפרט בשורה 1 של קוד הדוגמה אמור לחזור מהמתודה fun1 ערך בוליאני וניתן להניח שאכן ערך כזה חוזר. אין צורך לבדוק זאת).
- יורדו נקודות על אי <u>הגדרה מלאה</u> של DFA כפי שנלמד בתרגול.
- ניתן להשתמש בכתיב דוגמת $f \in C$ כדי לשאול אם למחלקה C יש מתודה בשם $f(\quad)$.
- ניתן להניח שישנן רק מחלקות A,B,C עם שמות המתודות הנתונות.
- ניתן להניח שאין קוד מת (קוד לא ישיג בזמן ריצה).
- כשייתכנו על אותו משתנה שתי קריאות ברצף למתודות ממחלקות שונות, על הDFA לדווח שגיאה.

3. (2 נק׳) האם הדיווח נאות (בכל קריאה של משתנה למתודה הטיפוס הדינמי של המשתנה מתאים אז

לא תדווח שגיאה)?
אם כן, הסבירו בקצרה למה. ואם לא, הראו מקרה קצר עליו הדיווח לא נאות והסבירו בקצרה מדוע במקרה זה השגיאה לא תדווח.

4. (2 נק') האם הדיווח שלם (אם יש קריאה של משתנה למתודה והטיפוס הדינמי של המשתנה לא מתאים אז תמיד תדווח שגיאה)?
אם כן, הסבירו בקצרה למה. ואם לא, הראו מקרה קצר עליו הדיווח לא שלם והסבירו במשפט או שניים למה במקרה זה השגיאה לא תדווח.

5. (4 נק') כזכור, ערכי החזרה של המתודות ידועים בפועל רק בזמן ריצה. עם זאת היינו רוצים במקרים מסוימים להצליח לזהות את ערכי החזרה כבר בזמן הקומפילציה.
הסבירו בקצרה איך מתוצאות סעיף 2 ניתן לפעמים לקבל את ערכי החזרה של המתודות של המחלקות השונות. ניתן להניח בסעיף זה שלמתודות בעלות אותו שם במחלקות שונות (לדוגמה fun2 ב-C וב-A) אותו ערך החזרה. התשובה צריכה להיות תאור אנליזה בתוך בלוק (ולא באמצעות DFA נוסף).

שאלה 4 (27 נק'): ניתוח תחבירי

1) (11 נק') נתון הדקדוק G הבא:

1) $S' \to S$
2) $S \to Aa$
3) $S \to Bb$
4) $A \to Ac$
5) $A \to \varepsilon$
6) $B \to Bc$
7) $B \to \varepsilon$

מעוניינים לבנות מנתח SLR עבור G (שימו לב שכבר הוספנו את כלל הגזירה $S' \to S$ לדקדוק).
כמו כן מעוניינים שהפרסר יבצע פעולת *reduce* ל A כאשר יש a בסוף הקלט ופעולת *reduce* ל B כאשר יש b בסוף הקלט.

א. (3 נק') מדוע רקורסיה שמאלית עדיפה על רקורסיה ימנית במנתח *shift-reduce*? הסבירו והדגימו.
ב. (2 נק') בנו את המצב הראשון של אוטומט המנתח והסבירו אילו קונפליקטים ישנם אשר גורמים לכך שהדקדוק לא *SLR*.
ג. (2 נק') האם הדקדוק G הוא *LR(1)*? הסבירו באמצעות אוטומט מתאים.
ד. (2 נק') הראו ששינוי כללי גזירה (4) ו (6) כך שיהיו עם רקורסיה ימנית פותרת את הבעיה במקרה זה.
ה. (2 נק') הסבירו את האינטואיציה של התוצאה מסעיף ד', מדוע השינוי הנ"ל פתר את הבעיה.
(מבחינת התנהגות המנתח והאינפורמציה הזמינה לו במהלך ניתוח לפי הדקדוק המקורי ובזה שלאחר השינוי).

2) (16 נק') במהלך ניסיון להוסיף מרקר לדקדוק ***LR(1)***, דנה נזכרה כי התנאי המאפשר להחליט האם הוספת המרקר תגרום לקונפליקט הוא כלהלן:
בהינתן הוספת מרקר ע"י החלפת כלל $A \to \alpha\beta$ בכלל: $A \to \alpha M\beta$ (והוספת המרקר $M \to \varepsilon$), יש לבדוק כי בכל מצב באוטומט בו יש פריט $[A \to \alpha \cdot \beta, t]$, אין פריט נוסף מהצורה: $[B \to \gamma \cdot \delta, r]$, כך ש $first(\beta t) \cap first(\delta r) \neq \phi$.

א. (4 נק׳) הוכיחו את נכונות התנאי הנ״ל.

ב. (8 נק׳) יוסי שם לב שניתן לעדן את התנאי באופן הבא: $first(\beta) \cap first(\delta) \neq \phi$.
הוכיחו או הפריכו את טענתו.
<u>רמז</u>: זכרו כי הדקדוק המקורי לפני הוספת המרקר הוא LR(1), כלומר חסר קונפליקטים.

ג. (4 נק׳) אם הדקדוק המקורי אינו בהכרח ב LR(1), האם יתכן שסוג הקונפליקט ישתנה בעקבות הוספת M?
הוכיחו תשובתכם.

<u>שאלה 5 (13 נקודות): Backpatching</u>

נתון מבנה בקרה חדש שהוא הרחבה של לולאת while:

$$S \rightarrow parallel - loop\ while\ (B_1)(B_2)\{S_1\}\{S_2\}$$

כאשר B גוזר ביטוי בוליאני.
<u>משמעות המבנה:</u>
ביצוע S_1 ו S_2 לפי התנאים B_1 וB_2. כלומר בכל איטרציה:
1. אםB_1 וגם B_2 מתקיימים יש לבצע את S_1 ואחריו את S_2.
2. אם B_1 לא מתקיים (ו B_2 מתקיים) יש לבצע את S_2 ובאיטרציות הבאות לא בודקים שוב את B_1 ולא מבצעים שוב את S_1.
3. כנ״ל לגבי המקרה ההפוך (כלומר B_2 מפסיק להתקיים ו B_1 ממשיך להתקיים) יש לבצע את S_1 כל עוד B_1 מתקיים ולא מבצעים את S_2.
4. אם שניהם לא מתקיימים, אז פשוט מסיימים את הלולאה.
<u>שימו לב:</u> לאחר ש B_1 הפסיק להתקיים, אין צורך לבדוק אותו שוב (וכנ״ל לגבי B_2).

הציעו פריסת קוד, המתאימה לשיטת backpatching, עבור מבנה הבקרה הנ״ל. על הקוד הנוצר להיות יעיל ככל האפשר. בנוסף, ציינו האם ואילו מרקרים יש להוסיף לכלל הגזירה לצורך המימוש בסכימת תרגום.

שימו לב: אין צורך לכתוב סכימת תרגום הממשת את פריסת הקוד בשאלה זו. עם זאת, יש להציג אך ורק פריסת קוד שניתן לממש אותה לפי התנאים של ההערות הנ״ל.

<u>הערות:</u>

1. אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
2. אין להשתמש במשתנים גלובליים בזמן קומפילציה.
3. המשתנים S,B הם המשתנים הסטנדרטיים המופיעים בדף הנוסחאות, ויש להם כללי גזירה בנוסף לכלל המופיע בשאלה.
4. אין להוסיף תכונות סמנטיות למשתנים B, S.
5. יש להשתמש אך ורק במרקרים M, N שנלמדו.

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
```

ניתוח סמנטי

אלגוריתם dfvisit לניתוח סמנטי עבור הגדרות L-attributed:

```
procedure dfvisit(node n) :
        foreach child m of n in left-to-right order do
                evaluate the inherited attributes of m
                dfvisit(m)
        end
        evaluate the synthesized attributes of n
```

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. פרמטרים וקריאה לפרוצדורות
```
param x
call p, n
return y
```
7. indexed assignments
```
x := y [ i ]
x [ i ] := y
```
8. השמה של כתובות ומצביעים
```
x := addr y
x := * y
* x := y
```

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:

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

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