שלבי קומפילציה

נתונה התוכנית הבאה ב-FanC

```
void do_move(int from, int to) { ... }
void move(int n, int from, int via, int to) {
    if (n > 1) {
      move(n - 1, from, to, via);
      do_move(from, to);
      move(n - 1, via, from, to);
    } else {
      do_move(from, to);
    }
}
void hanoi() {
    move(4, 1, 2, 3);
}
```

מתי תתרחש השגיאה?

1. מחליפים את האסימון הראשון בשורה 1 באסימון nothing.
   a. שגיאה בניתוח לקסיקלי
   b. אין שגיאה
   c. שגיאה בניתוח תחבירי
   d. שגיאה בניתוח סמנטי
   e. שגיאה בייצור קוד
   f. שגיאה בזמן ריצה

פתרון: nothing הוא אסימון חוקי (מסוג ID), אבל אין לו מקום לפי הדקדוק. לכן, שגיאה בניתוח תחבירי.

2. מוסיפים לסוף הפונקציה move את השורה "return n;" (אחרי שורה 9)
   a. שגיאה בניתוח סמנטי
   b. אין שגיאה

c. שגיאה בניתוח לקסיקלי
d. שגיאה בניתוח תחבירי
e. שגיאה בייצור קוד
f. שגיאה בזמן ריצה

פתרון: הפונקציה move היא מטיפוס void, ולכן לא אמור להיות לה return. השגיאה תהיה בשלב הניתוח הסמנטי.

3. מחליפים את שורה 12 בשורה: "move(2147483646, 1, 2, "3);
   a. שגיאה בזמן ריצה
   b. אין שגיאה
   c. שגיאה בניתוח לקסיקלי
   d. שגיאה בניתוח תחבירי
   e. שגיאה בניתוח סמנטי
   f. שגיאה בייצור קוד

פתרון: המספר 2147483646 הוא ממש ממש גדול, וגודל הרקורסיה הוא כגודל המספר הזה. לכן הקריאה הזו תוביל ל-stack overflow, וזו שגיאת זמן ריצה.

4. מסירים את הפסיק (",") בשורה 5.
   a. שגיאה בניתוח תחבירי
   b. אין שגיאה
   c. שגיאה בניתוח לקסיקלי
   d. שגיאה בניתוח סמנטי
   e. שגיאה בייצור קוד
   f. שגיאה בזמן ריצה

פתרון: הפסיק חייב להיות שם לפי הדקדוק, לכן זו שגיאה בניתוח תחבירי.

5. מחליפים את שורה 3 בשורה: "`if (n > 1) {`"
   a. אין שגיאה
   b. שגיאה בניתוח לקסיקלי
   c. שגיאה בניתוח תחבירי
   d. שגיאה בניתוח סמנטי
   e. שגיאה בייצור קוד
   f. שגיאה בזמן ריצה

פתרון: אין שגיאה. אומנם הפונקציה תפסיק לעשות את מה שהיא "אמורה" לעשות, אבל מבחינת הקומפילציה אין פה שגיאה.

6. אנחנו מעוניינים להוסיף לקומפיילר של FanC זיהוי אוטומטי ואופטימיזציה של רקורסיות זנב. רקורסיות זנב הן פונקציות רקורסיביות בהן הקריאה הרקורסיבית היא הפעולה האחרונה בכל מסלול בפונקציה, או לחילופין בסוף המסלול הפונקציה מחזירה ערך. בפונקציות מסוג זה אין צורך לשמור במחסנית את כל הקריאות הרקורסיביות, אלא רק את הנוכחית (למה?).
   מה השלב הקומפילציה המוקדם ביותר שבהכרח נצטרך לשנות כדי להוסיף את התמיכה הזו?
   a. ייצור קוד
   b. ניתוח לקסיקלי
   c. ניתוח תחבירי
   d. ניתוח סמנטי
   e. זמן ריצה

פתרון: אין צורך להוסיף או לשנות אסימונים (ניתוח לקסיקלי), את התחביר (ניתוח תחבירי), או את המשמעות של ביטויים (ניתוח סמנטי). מדובר באופטימזציה, אז השינוי יהיה בשלב ייצור הקוד.

7. נתונה הפונקציה הבאה:

```
int recursive_multiply(int x, int y, int acc) {
    if (x == 1)
        return y;
    return recursive_multiply(x - 1, y, acc + y);
}
void main() {
    recursive_multiply(2147483646, 1, 0);
}
```

בהינתן קומפיילר שתומך ברקורסיות זנב כאמור, האם ובאיזה שלב קומפילציה תתרחש שגיאה?

1. אין שגיאה
2. שגיאה בניתוח לקסיקלי
3. שגיאה בניתוח תחבירי
4. שגיאה בניתוח סמנטי
5. שגיאה בייצור קוד
6. שגיאה בזמן ריצה

פתרון: קומפיילר שתומך ברקורסיות זנב לא יצור דחיפות למחסנית וקריאות לפונק' עבור הקוד הזה, ויהפוך אותו ללולאה, כך שלא תהיה שגיאה.

8. תורם לפרוייקט LLVM הציע לשנות את רשומות ההפעלה שהקומפיילר מייצר כך שלא יכללו את כתובת החזרה. בהנחה ששפת המכונה לא קובעת את רשומת ההפעלה, בחר את התשובה הכי נכונה:
   a. ניתן לתמוך בחלק מהפונקציות כולל פונקציות רקורסיביות, אך לא ניתן לתמוך בפונקציות ספריה. הקובץ המקומפל יהיה גדול יותר.
   b. לא ניתן לתמוך בפונקציות בכלל.
   c. ניתן לתמוך בחלק מהפונקציות, אך לא ניתן לתמוך בפונקציות רקורסיביות או בפונקציות ספריה. הקובץ המקומפל יהיה גדול יותר.
   d. ניתן לתמוך בפונקציות מכל סוג. הקובץ המקומפל יהיה גדול יותר.

פתרון: ניתן לתמוך בפונקציות לא רקורסיביות: עבור כל מקום בו יש קריאה לפונקציה, נשכפל את הפונקציה עם המקום חזרה הזה (שקול ללעשות לפונקציה inlining.) לא ניתן לתמוך ברקורסיה במקרה הכללי, כי פונקציה רקורסיבית צריכה לחזור ללפחות 2 מקומות: למקום של הקריאה הרקורסיבית, ולמקום שבו הפונקציה נקראה מבחוץ. כמו כן, לא ניתן לתמוך בפונקציות ספריה, כי כל הפואנטה בפונקציות כאלה זה שהן כבר מקומפלות ונקראות ממספר תוכניות שונות.

נתונה התוכנית הבאה בשפת רביעיות:

```
a = 0
b = 1000
c = 0
if c >= b goto 13
d = c % 3
if d == 0 goto 10
e = c % 5
if e == 0 goto 10
goto 11
c + a = a
1 + c = c
goto 4
print a
```

1. רשמו את מספרי השורות של ה-leader-ים בתוכנית, וציירו את ה-CFG שלה.

2. בהינתן האופטימזציות copy propagation ו-constant folding, מה מבין המשפטים הבאים הוא הכי נכון?
   a. לפעמים נרצה לבצע copy propagation לפני constant folding, ולפעמים אחרי, לכן כדאי לבצע copy propagation גם לפני וגם אחרי constant folding.
   b. תמיד נרצה לבצע copy propagation לפני constant folding.
   c. תמיד נרצה לבצע copy propagation אחרי constant folding.
   d. הסדר הפעלה בין copy propagation ל-constant folding לא חשוב - אפשר לבצע את הפעולות בכל סדר ולקבל את אותן התוצאות.

פתרון: copy propagation מחליף אזכור של משתנה אחד במשתנה אחר, ו-constant folding פועל רק על קבועים. לכן הם פועלים על ביטויים שונים לחלוטין, ואין חשיבות לסדר ההפעלה שלהם.

דקדוקים

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

1. נתון דקדוק G=(V,T,P,S). נגדיר את הדקדוק G1 המתקבל מG להיות:
G1 = (V∪{S’}, T∪{t}, P∪{S’ → S, S’ → t}, S’) כאשר S’ ∉ V, t ∉ T. מה מהבאים הכי נכון:

א. תשובות ב' וג' נכונות.

ב. אם G שייך ל LL(1) אז G1 שייך ל LL(1).

ג. אם G שייך ל LR(1) אז G1 שייך ל LR(1).

ד. אף תשובה אינה נכונה.

פתרון:

אם G שייך ל LR(1) אז G1 שייך ל LR(1):

1. לאוטומט מתוספים שני מצבים, מצב אחד עם הפריט S’ → t•,$ בלבד ומצב אחד עם הפריט S’→ S•,$ בלבד. מצבים אלו אינם יכולים להכיל קונפליקט כי יש בהם רק פריט אחד ואין מהם קשתות יוצאות.
2. בנוסף, למצב התחילי מתוספים הפריטים S’ → •t,$ ו S’ → •S,$, שאינם יכולים לגרום לקונפליקט:

א. קונפליקט R/R לא יכול להיווצר כי הנקודה לא נמצאת בסוף הפריטים החדשים.

ב. הפריט S’ → •S,$ אינו גורם להוספת קשת, וכל הפריטים שנכנסים כתוצאה מ closure על פריט זה כבר היו במצב התחילי של האוטומט המקורי ולכן אינם גורמים לקונפליקט.

ג. הפריט S’ → •t,$ גורם להוספת קשת שמסומנת ב-t, אך קונפליקט S/R לא יכול להיווצר כי במצב התחילי אין אף פריט מהצורה α → A •,$. זאת מאחר שב lookahead של פריטי הכלל A→ α יכולים להופיע רק טרמינלים s עבורם קיים במצב פריט B → β • Aγ,r כך ש s ∈ first(γr) ; עבור הטרמינל t אין אף פריט כזה, כי t שייך רק ל first(t), ו first(S’) ו- S,’t אינם מופיעים אחרי אף משתנה בכללי הדקדוק (אלו משתנה וטרמינל חדשים) .

אם G שייך ל LL(1) אז G1 שייך ל LL(1):

1. לטבלת המנתח מתווספת שורה אחת עבור המשתנה S' ועמודה אחת עבור הטרמינל t.
2. בגלל שS' וt לא היו בדקדוק G, הטרמינל t לא שייך לfirst וfollow של אף אחד מן המשתנים בG, ולכן לא יהיה בselect של אף אחד מן הכללים בG. לכן בעמודה t קיים רק כלל הגזירה S’ → t.
3. בנוסף בשורת S' קיים רק הכלל S’ → S, והכלל S’ → t, אך הם לא נמצאים באותה המשבצת כי t לא נמצא בfirst או follow של S.
4. בשאר השורות והעמודות לא יהיה שינוי.

לכן אין אף קונפליקט בטבלת המנתח.

נתונים הדקדוקים הבאים:

G4:	G3:	G2:
S → A	S → AaA	S → BbB
A → Ba | bBb | Ab | a	A → AB | a	B → AB | b

	A → a	B → b	B → a

2. בוב בנה את האוטומט הפריפיקסי לפריטי LR(0) לדקדוקים הנ"ל וגילה שעבור בדיוק אחד מהדקדוקים מתקיימים שני הדברים הבאים:
   - קיים קונפליקט במנתח LR(0).
   - במנתח SLR לא קיימים קונפליקטים.

   מה המשפט הנכון ביותר:

א. בוב צודק, והדקדוק עליו הוא מדבר הוא G3.

ב. בוב צודק, והדקדוק עליו הוא מדבר הוא G2.

ג. בוב צודק, והדקדוק עליו הוא מדבר הוא G4.

ד. בוב טועה.

פתרון:

G2 הינו דקדוק LR(0) בניית הטבלה נטולת קונפליקטים.

G3 אינו דקדוק LR(0) כי קיים קונפליקט S/R באוטומט שנפתר באוטומט SLR באמצעות follow(S).

G4 אינו דקדוק בSLR כי R/R שאינו נפתר באוטומט SLR.

3. בוב נהנה לבנות את האוטומטים ולכן החליט לבנות גם את האוטומט הפריפיקסי לפריטי LR(1) לדקדוקים הנ"ל וגילה שבכל הדקדוקים, כל הקונפליקטים נפתרו. מה המשפט הנכון ביותר:

א. בוב צודק, כל הקונפליקטים נפתרו.

ב. בוב טועה, עדיין קיים קונפליקט R/R.

ג. בוב טועה, עדיין קיים קונפליקט S/R.

ד. בוב טועה, עדיין קיימים קונפליקט S/R וגם R/R.

פתרון:

לא קיימים קונפליקטים - הקונפליקט R/R בG4 נפתר במנתח LR(1).

4. בוב רוצה להשתמש בדקדוק G2 בשביל לנתח ביטויים בהם מספר הטרמינלים מסוג a גדול ממספר הטרמינלים מסוג b. (לדוגמה: הביטוי aaabbab יתקבל אצל בוב אך הביטוי abbaab לא יתקבל אצל בוב).

   בוב גילה שהוא יכול להוסיף בדיקות סמנטיות אך מותר לו להשתמש אך ורק בתכונות נורשות.

   בוב ביקש ממכם לייעץ לו כיצד לפתור את הבעיה. כיצד תייעצו לו לפעול?

   בחרו את העצה היעילה ביותר.

א. בוב יהיה חייב להשתמש גם בתכונות נוצרות ע"מ לבצע את בדיקותיו.

ב. בוב יכול לפתור את הבעיה עם הוספה של תכונה סמנטית אחת למשתנה B.

ג. בוב לא צריך להוסיף בדיקות סמנטיות כי כל הביטויים הנגזרים מהדקדוק הנ"ל עונים לדרישתו.

ד. בוב יכול לפתור את הבעיה עם הוספה של שתי תכונות סמנטיות למשתנה B.

ה. בוב יכול לפתור את הבעיה עם הוספה של תכונה סמנטית אחת למשתנה B ותכונה סמנטית אחת למשתנה A.

פתרון:

בכל מילה בדקדוק יש בדיוק 3 טרמינלים מסוג b. ע"מ לספור את מספר הטרמינלים מסוג a חייבים להשתמש בתכונות נוצרות, אחרת אין איך לדעת כמה פעמים נשתמש בכלל הגזירה $B \rightarrow AB$

אנליזה סטטית

בחברה שבה אתם עובדים מפתחים מחולל סיסמאות אוטומטי. המחולל מורכב מאוסף של פונקציות עבור סכימות שונות של מבנה הסיסמה, וכל פונקציה מורכבת ממשפטי השמה של תווים ומחרוזות מהצורה:

```
s = "xyz"
s = t
s = t ++ r
c = rand_char()
```

כאשר s, t, r הם משתנים מסוג מחרוזת, c הוא משתנה מסוג תו, ו ++ הוא אופרטור שרשור מחרוזות. rand_char היא פונקציה מובנית אשר מייצרת תו אקראי מתוך מרחב תווי ASCII הדפיסים. הקומפיילר שלכם מאפשר המרה אוטומטית של תו למחרוזת (באורך 1) כך שביטוי מהצורה s ++ c גם הוא תקין סמנטית.

כמו-כן קיימים בשפה תנאים בוליאניים מהצורה הבאה:

```
if (s == t) …
if (s != t) …
if (s.contains(c)) …
```

המתודה contains של טיפוס המחרוזת בודקת הימצאות מופע של תו נתון במחרוזת.

מדיניות האבטחה של החברה קובעת שעל סיסמה להכיל לפחות תו אחד מבין כל אחת מהקבוצות הבאות:

- אותיות קטנות (a-z)
- אותיות גדולות (A-Z)
- ספרות (0-9)
- סימנים מהקבוצה !@#$%^&*-|

א. הגדירו אנליזה סטטית שתבדוק האם הסיסמה המוחזרת מפונקציה עומדת בתנאי המדיניות. (כל פונקציה היא ישות עצמאית, ופונקציות לא יכולות לקרוא אחת לשנייה.)

<u>חשוב:</u> תנאי הנאותות הוא כזה שתוכנית מוגדרת כנכונה רק אם כל ריצה שלה מחזירה מחרוזת העומדת בתנאי. כתבו אנליזה מדויקת ככל האפשר, כלומר שתדחה כמה שפחות תוכניות נכונות.

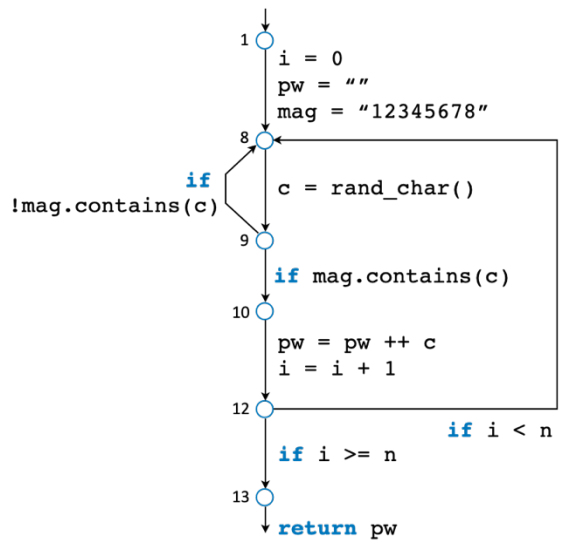
כתבו את ההגדרות בצורה גנרית, כך שיהיה קל להחליף את אוסף קבוצות התווים בקלות. כלומר, הגדירו אוסף של קבועים מתמטיים A1, A2, …, An שכל אחד מהם מייצג את אחת הקבוצות, ויישמו את ההגדרות ביחס לאוסף הזה.

1. הגדירו את הדומיין האבסטרקטי, את יחס הסדר בדומיין (⊑) ואת פעולת ה join (⊔).
2. הגדירו פונקציות אבסטרקציה (α) וקונקרטיזציה (γ) המקשרות בין הערכים האבסטרקטיים והקונקרטיים.
3. הגדירו את הסמנטיקה האבסטרקטית של ביטויים בשפה.
4. הגדירו את הסמנטיקה האבסטרקטית של משפטי תנאי.

<u>הנחה מקלה:</u> מותר להניח שהמחרוזת s בתנאים מהצורה s.contains(c) הינה **מחרוזת קבועה**, ושניתן לזהות זאת בעזרת אנליזת constant propagation כפי שנלמד בשיעור אופטימיזציות. (כמובן, שבביטויי תנאי שונים יכולים להופיע קבועים שונים.)

ב. הדגימו את ריצת האנליזה על ה CFG של התוכנית הבאה:

```
string spork(int n) {
  int i = 0;
  string pw = "";
  string mag = "12345678";
  do {
    char c;
    do {
      c = rand_char();
    } while (!mag.contains(c));
    s = s ++ c;
    i = i + 1;
  } while (i < n);
  return s;
}
```



x

הערות. (1) לולאות while .. do בודקות, כמקובל, את תנאי היציאה בסוף כל איטרציה במקום בתחילתה. (2) הניחו שמותר לאנליזה להתעלם מפעולות על משתנים מספריים ומערכי האמת של תנאים הכוללים מספרים, דוגמת המשפט i = i + 1 והתנאי i < n.

ציירו את ה CFG וכתבו על גבי הגרף או לצידו את הפתרון שמצאה האנליזה. **אין צורך** לפרט ערכי ביניים המתקבלים במהלך ריצת האנליזה.

ג. הסבירו כיצד הקומפיילר ישתמש בתוצאת האנליזה כדי להחליט אם לאשר את התוכנית או לדחות אותה.

האם התוכנית שבדוגמה עומדת במפרט המבוקש? האם תוצאת האנליזה מדויקת עבורה?

פתרון:

א. מגדירים את סריג קבוצת החזקה $L=P(\{A_1, A_2, \dots, A_n\})$ כאשר A_i מייצג את קבוצת התווים ה i. המשמעות של ערך בסריג הזה הוא קבוצה של קבוצות תווים אשר מכל אחת מהן מופיע לפחות תו אחד במחרוזת. למשל, אם למשתנה s משויך הערך $\{A_1, A_2\}$ הרי שהוא מכיל מחרוזת שיש בה לפחות אות קטנה אחת (A_1 בדוגמה) ואות גדולה אחת (A_2 בדוגמה). מכאן ואילך משתמשים רק ב A_i לתיאור הקבוצות וזה לא ישתנה אם נשנה את ההגדרות שלהם כקבוצות של תווים.

מכיוון שמדובר בבעיה מסוג **must** (כל ריצה אפשרית צריכה להחזיר מחרוזת העומדת בתנאים), נגדיר את יחס הסדר בסריג להיות $\sqsubseteq = \supseteq$ (הכלת קבוצות -- בכיוון ההפוך) ואז מקבלים כי $\sqcup = \cap$. כרגיל, בתוכנית עם k משתנים, מצב אבסטרקטי יתואר כאיבר בסריג החזקה L^k.

<u>אבסטרקציה וקונקרטיזציה</u>. מגדירים את פונקצית העזר β באופן הבא:

$$\beta(s) = \{A_i \mid \exists c. c \in s \wedge c \in A_i|\}$$

c) הוא תו אשר מצד אחד מופיע במחרוזת s ומצד שני שייך ל A_i.) מכאן קל להגדיר את α, γ באופן הבא:

$$\alpha(S) = \sqcup \{\beta(s) \mid s \in S|\}$$

$$\gamma(a) = \{s \mid \beta(s) \sqsubseteq a|\}$$

הביטויים בשפה הם "xyz..." (מחרוזת קבועה), s (משתנה מסוג מחרוזת), והרכבות עם שרשור e_1 ++ e_2. בנוסף ישנה הפונקציה המיוחדת rand_char.

$[\![w]\!]^{\#}\sigma^{\#} = \beta(w)$ (w is a constant)

$[\![s]\!]^{\#}\sigma^{\#} = \sigma^{\#}(s)$ (s is a string variable)

$[\![e_1 \text{ ++ } e_2]\!]^{\#}\sigma^{\#} = [\![e_1]\!]^{\#}\sigma^{\#} \cup [\![e_2]\!]^{\#}\sigma^{\#}$

$[\![\text{rand\_char}()]\!]^{\#}\sigma^{\#} = \top\ (= \emptyset)$

בנוגע לתנאים, $[\![\text{if} \ldots]\!]^{\#}$ מתאר את המעברים שמקיימים את התנאי בכל אחד מהמקרים. במקרה של שוויון, אפשר להבין שכל התווים שנמצאים באחד הצדדים נמצאים בשניהם. מאי-שוויון אי אפשר ללמוד כלום.

$[\![\text{if s == t}]\!]^{\#}\sigma^{\#} = \sigma^{\#}[s,t \mapsto [\![s]\!]^{\#}\sigma^{\#} \cup [\![t]\!]^{\#}\sigma^{\#}]$

$[\![\text{if s != t}]\!]^{\#}\sigma^{\#} = \sigma^{\#}$

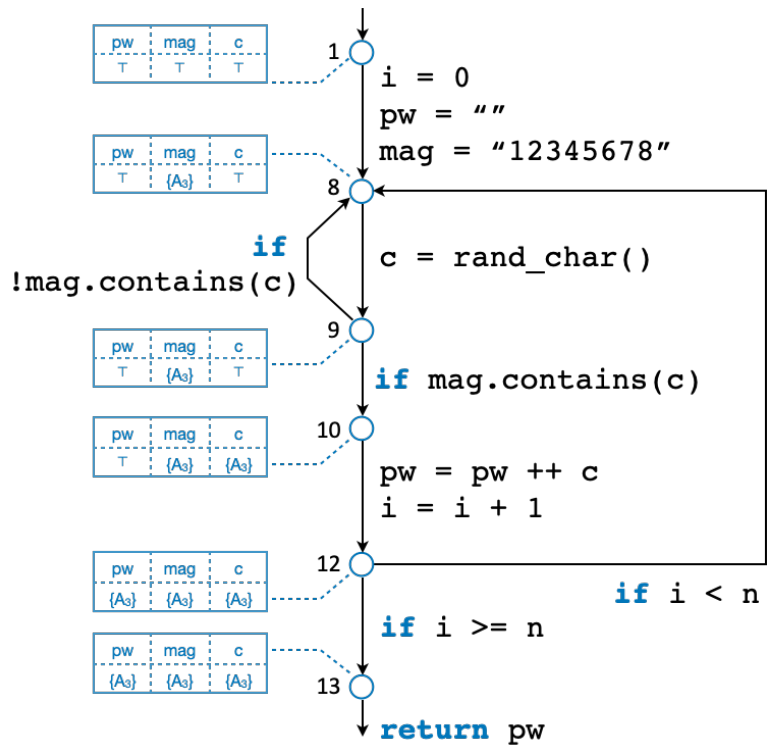
(הערה: במקרה הראשון אם אחד מצדדי השוויון אינו משתנה בודד, ההגדרה עובדת אותו דבר אבל מבלי לעדכן את הערך האבסטרקטי של אותו צד. עם זאת מותר להניח בה"כ שהשוויון הוא תמיד בין משתנים כי אפשר להעביר כל תוכנית לצורה כזו.)

במקרה של contains, לפי ההנחיה מותר להניח שהמחרוזת היא קבועה, w, ואז c יקבל את הערך A_i רק אם כל התווים במחרוזת שייכים ל A_i. נסמן chars(w) = קבוצת התווים שמופיעים במחרוזת w.

$[\![\text{if } w\text{.contains(c)}]\!]^{\#}\sigma^{\#} = \sigma^{\#}[c \mapsto \{A_i\}]$ if chars(w) $\subseteq A_i$, otherwise $\sigma^{\#}$

שימו לב שזה unsound להשתמש ב $[\![s]\!]^{\#}$ או ב $\beta(w)$ במקרה הזה מכיוון שישנם גם תווים שלא שייכים לאף A_i, והאבסטרקציה מתעלמת מהם; ואז קיימת אפשרות ש c שווה לאחד התווים האלה.

ב. בתוכנית הנתונה יש שלושה משתנים pw, mag, c (כלומר k=3). המצב האבסטרקטי בכל אחד מהמיקומים בגרף הבקרה מוצג בתרשים.



ג. בנקודה שבה מופיע משפט return e (לפני שחוזרים מהפונקציה) מוודאים שמתקיים

$[\![e]\!]^{\#}\sigma^{\#} = \bot$

זכרו ש $\bot$ הוא האיבר המינימלי בסריג, ובמקרה זה שווה ל $\{A_1, A_2, \dots, A_n\}$.

התוכנית שבסעיף ב׳ נדחית על ידי האנליזה מכיוון שהביטוי המוחזר מקבל את הערך האבסטרקטי $\{A_3\}$. ואכן, התוכנית אינה מחזירה (בהכרח) סיסמה חוקית — למעשה, אף ריצה שלה לא תחזיר סיסמה חוקית, כי כל הסיסמאות שמיוצרות בה מורכבות אך ורק מהספרות 1-8. לכן, האנליזה החזירה תוצאה מדויקת במקרה זה.

<u>נק׳)15: ייצור קוד (3שאלה</u>

בתוכנה שאתם מפתחים (לא זאת עם הסיסמאות, אחת אחרת) הוחלט לתת מענה לחריגות זמן ריצה בביטויים בוליאנים מסוימים, שיסומנו כ B'.

כעת, לכל ביטוי B', בנוסף ל falselist, truelist תתווסף התכונה exclist המכילה את רשימת הכתובות שמהן נקפוץ כשתתרחש חריגה (לצורך ביצוע backpatching עליהן).

נתון מבנה הבקרה הבא:

1) B -> B_list <u>with</u> S1

2) B_list -> B' <u>or</u> B_list1 | B'

3) B' -> true | false | error | ...

כאשר B' מייצג ביטויים בוליאניים שיכולים לזרוק חריגה. חריגה נזרקת כאשר משוערך הביטוי המיוחד error (מילה שמורה חדשה בשפה). בנוסף, B' מכיל את כל הביטויים הבוליאניים עם קשרים לוגיים ואופרטורי השוואה הרגילים.

הטרמינלים בדקדוק מסומנים בקו תחתון.

המבנה עובד באופן הבא:

1. כל עוד לא נזרקת חריגה בשום ביטוי בוליאני, B יתנהג כשרשרת or רגילה בין כל האיברים מסוג B'.
2. אם באחד הביטויים הבוליאנים נזרקת חריגה, אוסף הפקודות S1 יתבצעו ולאחר מכן החישוב יחזור אל הביטוי הבוליאני **הראשון** ברשימה, והחישוב ימשיך כרגיל.
3. הסמנטיקה של המבנה היא short-circuit.

דוגמא:

x = 0;

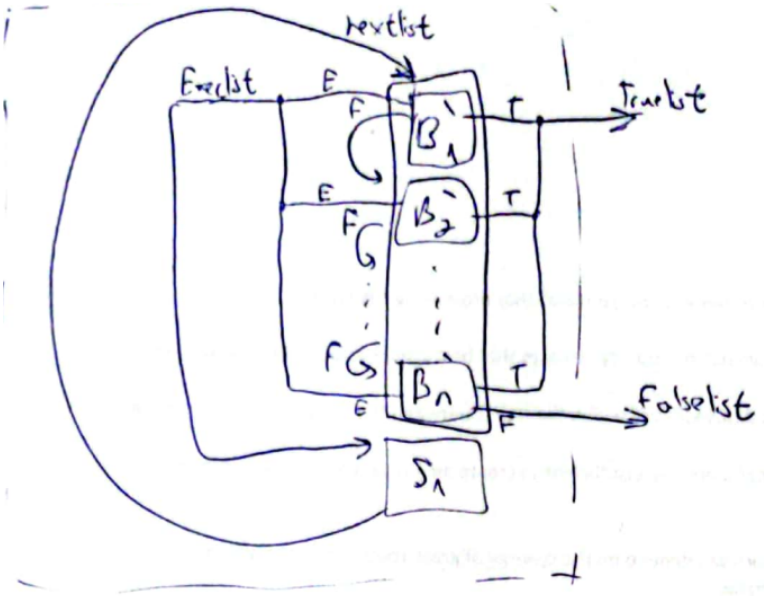
if (x!=0 or (x==0 and error) or 5/x != 5 with x = x+1;) print "True!";

כאשר הביטוי הבוליאני הראשון ישוערך לFalse, ואז הביטוי השני יזרוק חריגה.

נקדם את x ב-1, נחזור לתחילת הרשימה, ומכיוון שכעת הביטוי הראשון ישוערך לTrue, נצא ונדפיס True!.

<u>שימו לב.</u> אין שינוי בטיפול בשגיאות אריתמטיות ואחרות בחישוב ביטויים (למשל: חלוקה באפס, חריגה מגבולות מערך), אלה **אינן** זורקות חריגה אלא גורמות לשגיאה קריטית ולעצירת התוכנית, באופן הרגיל.

1. (5 נק') הציעו פריסת קוד לכללים 1,2 בשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. עבור כלל 3, הניחו כי כל הפעולות הסמנטיות עבור הכללים של B' שלא פורטו (המוצגים בדקדוק כ "...") כבר ממומשות ביחס ל short-circuiting.



2. (6 נק') כתבו סכימת תרגום לכללים 1,2 בשיטת backpatching המייצרת את פריסת הקוד שהצעתם בסעיף הקודם. על הסכימה להיות יעילה ככל האפשר, הן מבחינת זמן הריצה שלה והן מבחינת המקום בזיכרון הנדרש עבור התכונות הסמנטיות. כמו כן, הסבירו מהן התכונות שאתם משתמשים בהן עבור כל משתנה.

```
B_List -> MB'
{
B_List.start_quad = M.quad
B_List.falselist = B'.falselist
B_List.truelist = B'.truelist
B_List.execlist = B'.execlist
}
B_List -> MB' or B_list1
{
Backpatch(B'.falselist, B_list1.start_quad)
B_List.start_quad = M.quad
B_List.falselist = B_list1.falselist
B_List.truelist = merge(B_list1.truelist, B'.truelist)
```

```
B_List.execlist = merge(B_list1.execlist ,B'.execlist)
}
B -> B_list with M S1
{
Backpatch(B_list.execlist, M.quad)
Backpatch(S1.nextlist,B_list.start_quad)
B.truelist = B_list.truelist
B.falselist = B_list.falselist
}
```

3. (2 נק') כיצד פריסת הקוד הייתה משתנה אילו המבנה היה מוגדר ב eager evaluation? ניתן לתאר במילים.

נוסיף קטע קוד בinit (שנקפוץ אליו ממרקר N לפני B'), בו נאתחל משתנה t=0 למעקב אחרי הB' הנוכחי, וp=0 לתוצאה הסופית. לאחר מכן נקפיץ B' המתאים המתאים לפי t. אם t=n נקפוץ לfalselist או truelist בהתאם לp. Truelist של כל B' יבצע t=t+1 גם p=1, ו falselist של כל B' יבצע רק t=t+1. לאחר מכן נקפוץ לקטע הקוד שיעביר לB' הבא. נצטרך לשנות גם את S1.nextlist להצביע לinit.

בנוסף B_list יכיל בנוסף את הstart_quad של כל B' במחסנית.

4. (2 נק') כיצד פריסת הקוד הייתה משתנה אילו דרשנו שלאחר טיפול בחריגה כלשהי יש לחזור לחשב מאותו הביטוי הבוליאני שזרק את החריגה? ניתן לתאר במילים.

כמו בסעיף הקודם רק שהtruelist של כל B' לא מבצע את הקוד הנוסף אלא יוצא לtruelist של B, כדי לשמור על short-circuit.

אך כעת S1.nextlist לא יקפוץ לinit (כדי לא לאפס את t) אלא לקטע קוד לאחר מכן שקופץ לB' המתאים בהתאם לt.