2022-07-11

מבחן סוף סמסטר – מועד א'

טור א' (מספרי השאלות לטורים האחרים מופיעים ליד השאלות)

מרצה אחראית: ד"ר הילה פלג

מתרגלים: מתן פלד, תומר כהן, אלון קיטין, מתן עין-אבי

הוראות:

א. בטופס המבחן 14 עמודים, מתוכם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
ב. משך המבחן שלוש שעות (180 דקות).
ג. כל חומר עזר חיצוני אסור לשימוש.
ד. בשאלות הפתוחות, ניתן לציין לגבי סעיף או שאלה "לא יודע/ת". תשובה זו תזכה ב-20% מהניקוד. תשובות שגויות לא יזכו בניקוד.
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 הסגורות יש לסמן בטופס התשובות הנפרד בלבד. את התשובות לשאלות הפתוחות יש לכתוב במחברת הבחינה.
ח. ודאו כי אתם מגישים טופס תשובות ומחברת בחינה בלבד.

בהצלחה!

חלק א' - שאלות סגורות (50 נק')

טור א'

שלבי קומפילציה

נתון קטע הקוד הבא בשפת FanC:

```
1 : int myfunc(int u, int v) {
2 :   if (u < 0) u = -u;
3 :   if (v < 0) v = -v;
4 :   if (v > 0)
5 :     while ((u > 0) and (v > 0)) {
6 :         u = u - (u/v*v);
7 :         v = v - (v/u*u);
8 :     }
9 :   return u + v;
10:}
```

בסעיפים הבאים (שאלות 1-3) מוצגים שינויים (**בלתי תלויים**) לקוד של הקטע הרשום מעלה.
עבור כל שינוי ציינו את השלב המוקדם ביותר שבו נגלה את השגיאה (2 נק' לשאלה).

הנחיות:
1. ניתן להניח שאין שגיאות בקטע הקוד לפני השינוי
2. ניתן להתעלם משגיאות פוטנציאליות בשאר התוכנית

שאלה מספר 1: (5 נק')
(טור ב – שאלה 1, טור ג – שאלה 2)
לצורך מימוש לולאות for, איזה **שינוי** עלינו לעשות בשלב **הניתוח הסמנטי**?

א. הוספת token חדש
ב. הוספת ערכים מסוג חדש לטבלת הסימבולים
ג. הוספת בדיקות חדשות על שמות או טיפוסים
ד. הוספת כלל גזירה חדש
ה. לא נדרש שינוי בשלב הניתוח הסמנטי

מכיוון ששינוי הדקדוק המינימלי שניתן לבצע ישתמש בנונטרמינלים קיימים בדקדוק עבור statement ו-expression, השימוש בהם מכיל בדיקות תקינות עבור השימוש במשתנים

שאלה מספר 2: (2 נק')
(טור ב – שאלה 6, טור ג – שאלה 3)
נשנה את שם הפונקציה מ-myfunc ל-yourfunc.

א. שגיאה בייצור קוד
ב. שגיאה בניתוח תחבירי
ג. שגיאה בניתוח לקסיקלי
ד. שגיאה בזמן ריצה
ה. שגיאה בניתוח סמנטי
ו. אין שגיאה

מכיוון שאין שימוש ב-myfunc בתוך myfunc, ומכיוון שההנחיה היא להתעלם מכל שגיאה שעשויה להתרחש מחוץ לפונקציה, שינוי השם שלה ישנה את איך שהיא רשומה בטבלת הסמלים אך לא יגרום לשגיאה.

שאלה מספר 3: (2 נק')
(טור ב' – שאלה 3, טור ג – שאלה 4)
קיים בקומפיילר באג הגורם לו להדפיס שגיאה בשורה 6 הטוענת ש-v אינו מוגדר. למרבה הפלא, אם מחליפים את הסדר של שורות 6 ו-7 אז הקומפיילר מדפיס שגיאה בשורה 7 הטוענת ש-v אינו מוגדר. באיזה שלב קומפילציה מתחבא הבאג?

א. ניתוח לקסיקלי
ב. זמן ריצה
ג. ניתוח סמנטי
ד. ייצור קוד
ה. ניתוח תחבירי

אם אחרי החלפת סדר השורות הבאג גם החליף את השורה שמעלה שגיאה, אנחנו לומדים שזה לא יכול להיות בניתוח לקסיקלי (כי ה-token-ים היו מתחלפים גם), ניתוח תחבירי (כי מדובר על הסדר בין שני משפטים שונים),וייצור קוד וזמן ריצה לא הגיוניים בהקשר הזה. לכן מדובר על ניתוח סמנטי, כי רק שם יש משמעות לשמות של המשתנים (ההבדל ביחיד בין השורות הללו).

שאלה מספר 4: (5 נק')
(טור ב – שאלה 4, טור ג – שאלה 7)
נרצה להוסיף לשפת FanC תמיכה בלולאות for, אבל עדיין נשתמש במילה השמורה while בשביל הלולאות הללו. נוכל להבדיל בין הסוגים השונים של הלולאות לפי קיומן של סימני נקודה-פסיק (;) בתנאי הלולאה.

מהו שלב הקומפילציה המאוחר ביותר אותו **לא** נצטרך לשנות?

א. ניתוח לקסיקלי
ב. ייצור קוד
ג. ניתוח תחבירי
ד. ניתוח סמנטי
ה. זמן ריצה

מכיוון שאנחנו ממחזרים את האסימון while לצורך הלולאה החדשה, אין צורך לשנות דבר בשלב הלקסיקלי. כן נצטרך להוסיף כלל דקדוק חדש (תחבירי), שלא ידרוש בדיקות סמנטיות חדשות שכן כולן מובנות בתתי הכללים הקיימים והבדיקה כי התנאי בוליאני כבר קיימת ב-while, שלב ייצור הקוד יצטרך לייצר את מבנה הקפיצות של לולאת for, ובזמן הריצה לא ישתנה דבר.
לפי כוונת השאלה – השלב האחרון שלא שונה לפני השינוי הראשון, התשובה היא שלב הניתוח הלקסיקלי. בנוסף נקבל גם את השלב האחרון ממש שאין בו שינוי : זמן ריצה.

שאלה מספר 5: (2 נק')
(טור ב – שאלה 5, טור ג – שאלה 6)
באיזה שלב מתבצעת ההחלטה האם קודם לחלק את u ב-v או להכפיל את v ב-v בשורה 6?

א. ניתוח סמנטי
ב. זמן ריצה
ג. ניתוח תחבירי
ד. ייצור קוד
ה. ניתוח לקסיקלי

קדימות אופרטורים תכריע את ההחלטה בזמן גזירת התכנית בניתוח התחבירי.

שאלה מספר 6: (2 נק')
(טור ב – שאלה 2, טור ג – שאלה 5)
נחליף את הסימן '-' בשורה 2 בסימן '~'.

א. שגיאה בניתוח סמנטי
ב. שגיאה בזמן ריצה
ג. שגיאה בניתוח לקסיקלי
ד. שגיאה בניתוח תחבירי
ה. אין שגיאה
ו. שגיאה בייצור קוד

אין בשפת FanC אסימון המכיל (או מתחיל ב-) ~, ולכן בהגיעו לתו ~ המנתח הלקסיקלי לא יוכל להתקדם.

שאלה מספר 7: (2 נק')
(טור ב – שאלה 7, טור ג – שאלה 1)
נמחק את כל הרווחים (whitespace) הקיימים בתוכנית.

א. שגיאה בניתוח סמנטי
ב. שגיאה בניתוח תחבירי
ג. שגיאה בייצור קוד
ד. אין שגיאה
ה. שגיאה בזמן ריצה
ו. שגיאה בניתוח לקסיקלי

מחיקת הרווחים תגרום למילים שמורות "להימעך" לכדי רצפי אותיות שיזוהו כאסימון ID, למשל intmyfunc או intu. אבל בגלל היעלמות המילים השמורות, בשלב הניתוח התחבירי, לא יימצא כלל מתאים כדי לגזור את רצף האסימונים.

אופטימיזציות

נתון הקוד הבא בשפת ביניים, המייצג את הגירסה המקומפלת (ע"י קומפיילר תקין) של myfunc מהשאלה הקודמת. הניחו כי u ו-v הם פרמטרים, וכי return מתבצע ע"י השמה למשתנה ששמו כשם הפונקציה.

```
1:    if u >= 0 goto 3
2:    u = 0 - u
3:    if v >= 0 goto 5
4:    v = 0 - u
5:    if v <= 0 goto 18
6:    tmp1 = u / v
7:    tmp2 = tmp1 * v
8:    u = u - tmp2
9:    tmp3 = v / u
10:   tmp4 = tmp3 * u
11:   goto 12
12:   tmp5 = v - tmp4
13:   v = 0 + tmp5
14:   if u <= 0 goto 6
15:   goto 16
16:   if v <= 0 goto 6
17:   tmp6 = 0
18:   tmp7 = u + v
19:   myfunc = tmp7 + tmp6
```

שאלה מספר 8: (5 נק')

מי מבין הבאים **אינו** basic block ב-CFG שנוצר מהקוד?

א.

```
tmp6 = 0
tmp7 = u + v
myfunc =tmp7 + tmp6
```

ב.

```
u = 0 - u
```

ג.

```
tmp5 = v - tmp4
v = 0 + tmp5
if u <= 0 goto 6
```

ד.

```
goto 16
```

בגלל הקפיצה בשורה 5, שורה 18 היא leader, והשורות בא' יפוצלו לשני בלוקים.

שאלה מספר 9: (5 נק')
ביצענו על הקוד את כל האופטימיזציות שלמדנו בכיתה מלבד branch chaining.
איזו טענה מבין הבאות נכונה?

א. ביצוע branch chaining לפני האופטימיזציות האחרות יוביל לקוד יעיל יותר מאשר ביצוע branch chaining בסוף.
ב. מספר ה-basic blocks ב-CFG נשאר זהה.
ג. אף אחת מהתשובות אינה נכטונה
ד. הקוד שהתקבל אחרי ביצוע האופטימיזציות זהה לקוד ההתחלתי.
ה. ביצוע branch chaining על הקוד שהתקבל אחרי ביצוע כל האופטימיזציות לא יוביל לשינוי נוסף.

ה-branch chaining בדוגמא זו אינו תלוי באופטימיזציות אחרות, אז א' אינו נכון. יש אופטימיזציות שאפשר לבצע על הקוד, למשל constant propagation בשורה 19, ולכן ד' אינו נכון. בלי לבצע branch chaining, מספר ה- basic blocks נשאר זהה מכיוון שכל ה-goto-ים נשארו באותו מקום, לכן ב' נכון. בשורות 11 ו-15 יש goto לשורה העוקבת; זו דוגמא שונה מספיק מזו שראינו בכיתה כך שהחלטנו לקבל גם את ה'.

דקדוקים

שאלה מספר 10: (4 נק')
יהי G דקדוק של שפה סופית L. סמנו את הטענה הנכונה ביותר:

א. G לאו דווקא ב-LR(0) אבל ניתן לבנות מנתח LR(0) לדקדוק כלשהו שמקבל את L.
ב. G לאו דווקא ב-LR(0) ולא ניתן לבנות מנתח LR(0) לאף דקדוק שמקבל L.
ג. G בהכרח ב-LR(0) ולכן ניתן לבנות מנתח LR(0) לדקדוק שמקבל את L.

אף תשובה אינה נכונה: G לאו דווקא ב-LR(0) כי קל לייצר דקדוק לא חד-משמעי גם לשפה סופית. אך ניתן לבחור שפה סופית כך שלא ניתן לבנות לה מנתח LR(0) כיוון שמילה אחת בשפה היא prefix של מילה אחרת, למשל השפה {a, aa}. כן ניתן לבנות מנתח SLR. השאלה נזרקה.

יהי G_1 הדקדוק הבא:

$S \rightarrow A\ a$
$S \rightarrow b\ A\ c$
$S \rightarrow d\ c$
$S \rightarrow b\ d\ a$
$A \rightarrow d$

שאלה מספר 11: (3 נק')
(טור ב – שאלה 13, טור ג – שאלה 11)
באוטומט הפרפיקסי למנתח LR(0) עבור G_1, דרגת היציאה (מספר הקשתות היוצאות) המקסימלית של מצב כלשהו

א. שווה ל 4
ב. גדולה מ-4
ג. שווה ל 3
ד. שווה ל 2
ה. שווה ל 1

כאשר בונים את האוטומט של G1, ראשית נוסיף כלל התחלתי חדש $S' \to S$. מכיוון שאין רקורסיה המשתמשת ב-S אין צורך להוסיף $ לסוף הכלל ההתחלתי כי אין צורך לספק לרקורסיה נקודת עצירה, אבל בניית האוטומט הפרפיקסי מתחילה מכלל התחלתי בודד והפעלת סגור עליו, ומכיוון של-S יש מספר כללים יש לעטוף אותו בכלל בודד עם משתנה חדש. מכאן, למצב ההתחלתי באוטומט יהיו 4 קשתות יוצאות, עבור S, A, b, ו-d, וזהו המצב עם דרגת היציאה המקסימלית.

שאלה מספר 12: (3 נק')
(טור ב – שאלה 11, טור ג – שאלה 13)
בטבלת הניתוח למנתח SLR עבור $\underline{G_1}$, מספר השורות שיש בהן קונפליקט הוא:

א. 2
ב. 0
ג. 3
ד. 1

מכיוון שגם a וגם c ב-FOLLOW של A, בשתי השורות בהן מופיע רידוס עבור $A \to d$ יהיה קונפליקט.

שאלה מספר 13: (3 נק')
(טור ב – שאלה 12, טור ג – שאלה 12)
בטבלת הניתוח למנתח LR(0) עבור G_1, מספר השורות שיש בהן קונפליקט הוא:

א. 0
ב. 2
ג. 1
ד. 3

בשתי השורות בהן מופיע רידוס עבור $A \to d$ יהיה קונפליקט.

יהי $\underline{G_2}$ הדקדוק הבא:
$A \to a\,A\,a$
$A \to \varepsilon$

שאלה מספר 14: (4 נק')
סמנו את הטענה החזקה ביותר לגבי הדקדוק $\underline{G_2}$

א. הדקדוק ב SLR
ב. הדקדוק ב LR(0)
ג. הדקדוק ב LR(1)
ד. אף אחת מן התשובות אינה נכונה

בטבלת הניתוח LR(1) של הדקדוק ישנם שני קונפליקטי S/R בין הכלל $A \to \varepsilon$ לאסימון a. מכיוון שזו המחלקה החזקה ביותר, מכאן הדקדוק גם אינו באף אחת מהמחלקות הנוספות.

<u>שאלה מספר 15:</u> (3 נק')

לכל דקדוק G_3 המקבל את השפה של G_2 מתקיים : G_2 ב-LR(1) **אם ורק אם** G_3 ב-LR(1)

א. לא נכון
ב. נכון
ג. לא ניתן להוכיח

הדקדוק G2 אינו ב-LR(1). מכאן, כדי שיתקיים האמ"מ צריך להראות כי לכל G3 המקבל את L(G2), G3 גם אינו ב-LR(1). השפה של G2 היא מספר זוגי של a, ולכן ניתן לקבל אותה גם עם הדקדוק $\boldsymbol{A \rightarrow A\,a\,a \mid \varepsilon}$ שכן נמצא ב-LR(1). מכיוון שיש דקדוק כזה, האמ"מ לא מתקיים.

חלק ב' - שאלות פתוחות (50 נק')

שאלה 1 : ייצור קוד (20 נק')

תומר1 ותומר2 הם סטודנטים למדמ"ח הרשומים הקורס מבוא לתורת הקומפילציה. לאחר שהגישו את תרגיל הבית החמישי (בזמן) הם החליטו ליצור שפה חדשה בשם FanCier שתממש גם את מבנה הבקרה for משום שהרגישו אכזבה שלא מימשו אותו בשפת ה-FanC.
אבל, תומר1 לא יודע מתי לעצור והחליט להעצים את היכולות של מבנה הבקרה באופן הבא :

```
fancy_for(4<5, int i=0, int j=7; i<3, j>2; i++, j--){

    print("%d,%d=",i,j);

}//prints 0,7=1,6=2,5=
```

בתוך הסוגריים של fancy_for יש 3 רשימות לא ריקות של statements וגם Boolean expressions. הקוד עובר באופן short circuit על הרשימות, מבצע כל פקודה שהוא מגיע אליה ובודק כל תנאי. במידה והתנאי נכון, ממשיכים לאיבר הבא ברשימה, אחרת יוצאים מהלולאה. הרשימה הראשונה מורצת רק לפני האיטרציה הראשונה, הרשימה השנייה מורצת לפני כל איטרציה והרשימה השלישית מורצת אחרי כל איטרציה.

$S \rightarrow$ ***fancy_for*** $(SB\_LIST_1 ; SB\_LIST_2 ; SB\_LIST_3) \{ S \}$

$SB\_LIST \rightarrow SB\_LIST_1 , S$

$SB\_LIST \rightarrow SB\_LIST_1 , B$

$SB\_LIST \rightarrow S$

$SB\_LIST \rightarrow B$

- ניתן להניח שבקוד לא יהיו שגיאות קומפילציה
- אין לשנות את הדקדוק פרט למרקרים M,N.
- אין חשיבות לסקופים
- אסור להשתמש במשתנים גלובליים
- למשתנה S יש עוד כללי גזירה ו**תמיד** יכיל nextlist

א. (6 נק') הציעו לתומר1 פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות הסמנטיות שאתם משתמשים בהן עבור כל 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.

לאחר השינויים הנ"ל, מתן, שותף של תומר2, רוצה שהקוד בתוך הלולאה יורץ במידה ואחד התנאים הבוליאניים ברשימה השנייה מתקיים (תוך דילוג על ה-statements שמופיעים בהמשך אותה הרשימה).

ג. עזרו למתן לממש את השינוי, יש לתאר במילים אילו שינויים יש להכניס לתשובות של סעיפים א, ב.

שאלה 2 : אנליזה סטטית (30 נק')

סטודנטים בפרוייקט בקומפילציה מ' מימשו שפת תכנות חדשה שהיא גרסה של פייתון אבל statically typed.
השפה מכילה משתנים מטיפוס `str` המייצג מחרוזת מעל האי"ב המכיל את התווים *a-z*, *A-Z*, *0-9*. אין חשיבות לאופן בו המחרוזת מיוצגת (למשל, אין צורך להתייחס לפרטי מימוש כמו terminating null, רק לתווים בתוך המחרוזת). על המחרוזות בשפה ניתן לבצע את הפעולות הבאות:

`'str'`	הגדרת קבוע מחרוזת
`x + y`	שרשור שתי המחרוזות *x* ו-*y*
`x * n`	שרשור של המחרוזת *x* עם עצמה *n* פעמים. אם *n<=0* תוחזר מחרוזת ריקה.
`x.replace(y,z)`	יחזיר מחרוזת שהיא העתק של *x* בה כל מופע של רצף התווים *y* יוחלף ברצף התווים *z*
`x.upper()`	יחזיר מחרוזת שהיא העתק של *x* בה כל תו בין *a* ל-*z* יוחלף בתו הגדול המתאים (בין *A* ל-*Z*)
`x.lower()`	יחזיר מחרוזת שהיא העתק של *x* בה כל תו בין *A* ל-*Z* יוחלף בתו הקטן המתאים (בין *a* ל-*z*)
`x.find(y)`	יחזיר את המיקום הראשון של *y* בתוך *x*. אם *y* אינו תת-מחרוזת של *x*, יחזיר *-1*

ניתן לבצע השמה מכל ביטוי מטיפוס `str` למשתנה מטיפוס `str`. כל המחרוזות בשפה הן אינן ניתנות לשינוי (immutable) וכל פעולה המייצרת מחרוזת שונה תייצר העתק שלה.

הסטודנטים רוצים לאפשר למתכנתים בשפה שלהם לבצע אימות של assertions בשפה, ולשם כך לקחו תכנית מייצגת. לפעימה הראשונה של הפרוייקט הם התחייבו לבצע אנליזה שתעזור להם לבדוק **את ה-assert בשורה 9**.

```
1: def bar(str x, int n):

2:     str y = ''
3:     while (n > 0):
4:         n = n-1
5:         if n != 0:
6:             y = x.replace('s','t')
7:         else:
8:             y = (x.upper() + 'w') * 3
9:     assert(y.find('s') == -1)
10:    assert(y.find('w') != -1)
```

א. (15 נק') עזרו לסטודנטים להגיש את הפרוייקט בזמן והגדירו אנליזה שתעזור להם לבדוק את ה-assert בשורה 9.

1. הסבירו בקצרה(!) מה נדרש על מנת לבדוק את ה-assert בשורה 9 (גם אם זה לא יספיק כדי לבדוק את ה-assert בשורה 10)

האסרט בשורה 9 בודק האם תו כלשהו (במקרה הזה ספציפית s) לא נמצא במחרוזת. כדי לבדוק אותו, צריך לעקוב אחרי המחרוזת ולבדוק האם קיים מסלול כלשהו שבו עלול להיות בה s.

2. הגדירו את הדומיין האבסטרקטי, את יחס הסדר בדומיין (⊑) ואת פעולת ה-join (⊔).

נגדיר את קב' התווים המותרים במחרוזת Σ, ונבחר את הדומיין שלנו להיות $\mathcal{P}(\Sigma)$ כאשר ⊑=⊆ ⊔=∪. בדומיין הזה, קבוצת תווים a תציין כי כל תו ב-a עשוי להיות במחרוזת. כרגע נגדיר דומיין שמייצג ערכי מחרוזת, בלי קשר למשתנים, ובסעיף ב' נשתמש בסריג המכפלה כדי לייצג את ערכי המשתנים בתכנית.

פתרונות אלטרנטיביים קבילים: עבור מי שרוצה לבדוק אך ורק את שורה 9 ולהתעלם מכל תו בודד אחר ב-find, גם התשובה $\mathcal{P}(\{s\})$ עם הכלה ואיחוד תספיק, ותייצר דומיין עם שני איברים - $\{s\}$, ∅ המציינים כי עשוי להיות s במחרוזת או בוודאות אין s במחרוזת.
בנוסף, ניתן גם לעבוד הפוך – להגדיר את ⊔=∩, ⊑=⊇, ולהגדיר את הדומיין כך שישמור את התווים שבוודאות לא מוכלים במחרוזת, תשובה שתעבוד גם עם קבוצת החזקה של כל האי"ב וגם

עם קבוצת החזקה של $\{s\}$. במקרה הזה, צריך להפוך את פונקציות האבסטרקציה\קונקרטיזציה, כיוון פעולות הקבוצות גם בכל הגדרות הסמנטיקה בהתאם.

3. הגדירו פונקציות אבסטרקציה (α) וקונקרטיזציה (γ) המקשרות בין הערכים האבסטרקטיים והקונקרטיים.

נגדיר את פונקציית האבסטרקציה לאחד את כל התווים שמופיעים באיזושהי מחרוזת מתוך קבוצת מחרוזות:

$$\alpha(S) = \bigcup_{s \in S} \{c | c \in s\}$$

ואת פונקציית הקונקרטיזציה להרכיב את כל המחרוזות האפשריות מקבוצת התווים הנתונה ע"י הביטוי הרגולרי $\gamma(a) = a *$

4. הגדירו את הסמנטיקה האבסטרקטית של כל הביטויים בשפה **למעט** של `find`.

<u>הנחה מקלה:</u> ניתן להניח כי יש לכם את תוצאת הריצה של constant propagation על מחרוזות זמינה לשימושכם. מותר לפצל את הטיפול בפעולות למקרים לפיה במידת הצורך.

$$[\!['str']\!]^{\#}\sigma^{\#} = \alpha(\{'str'\}) = \{c | \ c \in 'str'\}$$

מחרוזת קבועה תהפוך לאבסטרקציה של מחרוזת קבועה. מי שהגדיר פונקציית β, כאן היה מאוד נוח להשתמש בה.

$$[\![x \ + \ y]\!]^{\#}\sigma^{\#} = [\![x]\!]^{\#}\sigma^{\#} \cup [\![y]\!]^{\#}\sigma^{\#}$$

בשרשור של שתי מחרוזות עשויים להיות כל התווים שעשויים להיות בראשונה וגם כל התווים שעשויים להיות בשניה. מכיוון ש-x ו-y לא חייבים להיות משתנים אלא יכולים להיות קבועים (כמו בתכנית) או ביטויים מורכבים אחרים מסוג מחרוזת, השימוש כאן אינו ב-$\sigma^{\#}(x)$ אלא בשערוך הרקורסיבי של x.

$$[\![x \ * \ n]\!]^{\#}\sigma^{\#} = [\![x]\!]^{\#}$$

מכיוון ש-n הוא ערך מספרי ואנחנו לא מתעניינים בערכים מספריים, אנחנו מחפשים את הגרסה הקונסרבטיביות ביותר שתהיה נכונה והיא להשאיר את כל התווים שהיו עשויים להיות ב-x. כך אם n הוא במקרה מספר חיובי אז דייקנו לגמרי, ואם n הוא אפס או מספר שלילי, אז המחרוזת תתאפס אבל מחרוזת ריקה מיוצגת על ידי הקונקרטיזציה של כל איבר בדומיין שלנו ולכן אנחנו מייצגים את כל המצבים הישיגים והכול טוב.

אלה מכם שהניחו את קיומו של *constant propagation* על ערכים מספריים ולא רק על מחרוזות יכלו לפצל כאן לשני מקרים: כאשר n קבוע וגם ערכו קטן או שווה ל-0, $[\![x \ * \ n]\!]^{\#}\sigma^{\#} = \emptyset$ ובכל מקרה אחר, כפי שהוגדר לעיל.

$$[\![x.replace(y,z)]\!]^{\#}\sigma^{\#} = \begin{cases} [\![x]\!]^{\#}\sigma^{\#} \setminus [\![y]\!]^{\#}\sigma^{\#} \cup [\![z]\!]^{\#}\sigma^{\#} & y \ is \ constant, len(y) = 1 \\ [\![x]\!]^{\#}\sigma^{\#} & [\![x]\!]^{\#}\sigma^{\#} \cap [\![y]\!]^{\#}\sigma^{\#} = \emptyset \\ [\![x]\!]^{\#}\sigma^{\#} \cup [\![z]\!]^{\#}\sigma^{\#} & \text{אחרת} \end{cases}$$

חשוב לשים לב שבגלל שאיבדנו מידע בתהליך האבסטרקציה, להשתמש תמיד באופציה הנכונה פשוט לא יהיה נכון. ניקח לדוגמה את הקוד `'s'.replace('sss','q')`. אחרי הפירוק הרקורסיבי לשערוך הארגומנטים נקבל בעצם $\{'s'\}.replace(\{'s'\},\{'q'\})$, ואם לא נשתמש בתוצאת *constant propagation* ונפעיל מיד את השורה הראשונה נקבל את הערך האבסטרקטי $\{'q'\}$, בעוד ששיערוך לפי הסמנטיקה הקונקרטית יחזיר את המחרוזת *'s'* – כלומר קיבלנו מצב אבסטרקטי שלא מכסה את כל המצבים הישיגים!

למי שהתעצל לפצל למקרים, $[\![x]\!]^{\#}\sigma^{\#} \cup [\![z]\!]^{\#}\sigma^{\#}$ מכסה את כל המצבים הישיגים בלי תוצאות *constant propagation* ורק מאבד דיוק בדרך. כמובן, אם עשיתם זאת, לא תצליחו לבדוק את ה-*assert*...

$$[\![x.upper(\ \ )]\!]^{\#}\sigma^{\#} = \{upper(c) \mid c \in [\![x]\!]^{\#}\sigma^{\#}\}$$

כלומר עבור כל תו ב-x, נחליף אותו באות גדולה אם הוא אות קטנה, נשאיר אותו אותו הדבר אם הוא ספרה או אות גדולה. חשוב לשים לב שלא מספיק לחסר מהקבוצה את כל האותיות הקטנות, צריך להוסיף גם את האותיות הגדולות המקבילות! לחלופין, ניתן להתעצל ולבצע *overapproximation* שתיקח את כל האותיות הגדולות וכל הספרות בלבד. זה מאוד מאבד דיוק, אבל במקרה מספיק לתכנית הנוכחית.

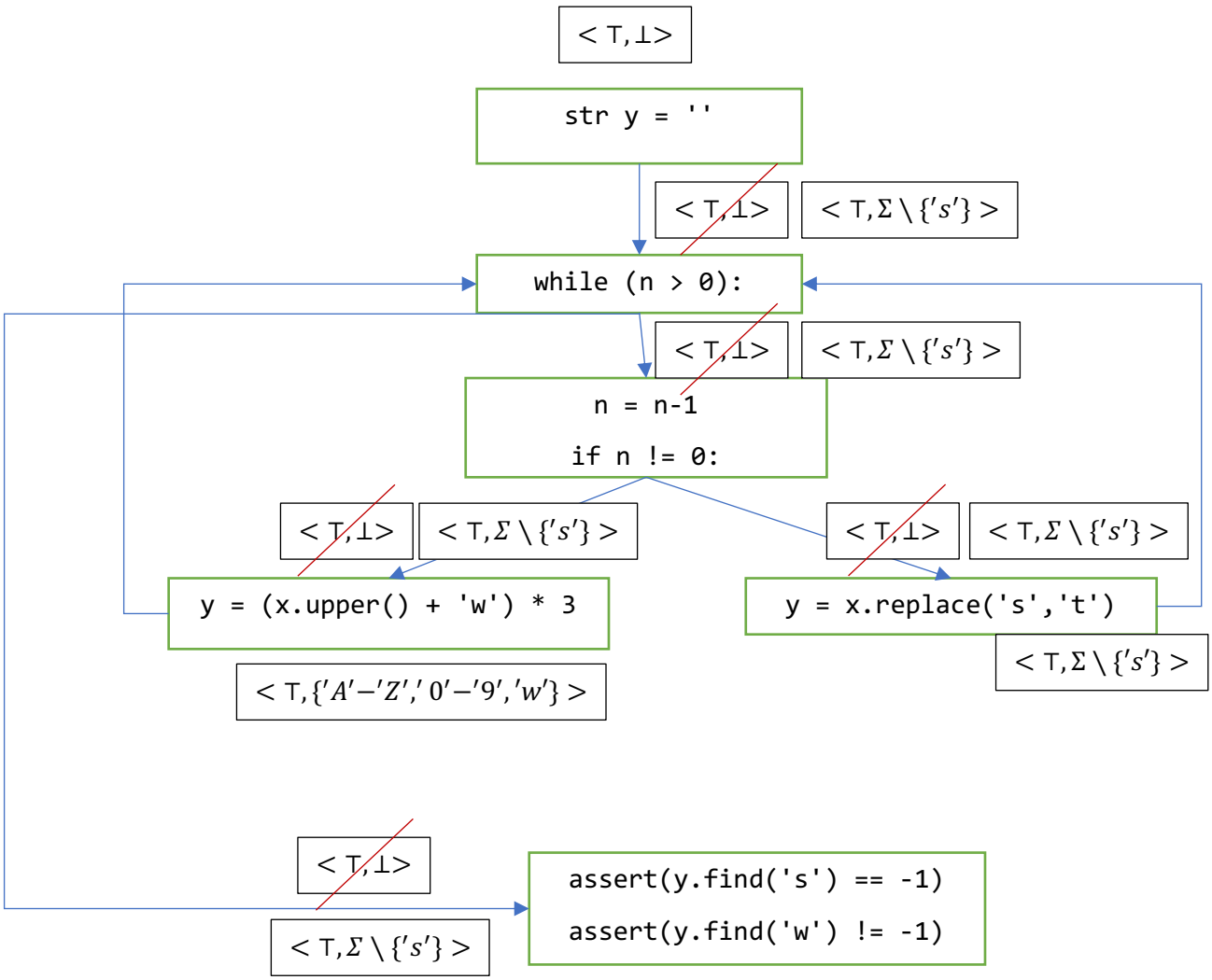
$$[\![x.lower(\ \ )]\!]^{\#}\sigma^{\#} = \{lower(c) \mid c \in [\![x]\!]^{\#}\sigma^{\#}\}$$

ואותו הדבר עבור *lower*.

ב. (10 נק') הריצו את האנליזה.

1. ציירו את ה-CFG של הפונקציה bar.
2. השתמשו באנליזה שלכם כדי לנתח את הפונקציה.

הנחות מקלות: אין צורך להתייחס כלל למשתנים מספריים, ואין צורך להתייחס לתוכן של תנאים בתכנית. השתמשו בפונקציות המעברים הסטנדרטיות ובהגדרה הרקורסיבית של חישוב ביטוי כפי שנלמדו בכיתה.



נשתמש בסריג המכפלה כדי לייצג את שני משתני המחרוזות, x ו-y. נאתחל את האנליזה שלנו עם ערך לפני אתחול (בוטום) עבור y ומכיוון ש-x הוא פרמטר לפונקציה ערכו יכול להיות כל דבר ולכן נאתחל אותו עם טופ. נשתמש בסמנטיקה שראינו בשיעור עבור משפטי השמה, ונבצע את ההשמה של הערך האבסטרקטי החדש של y בשורה 2 – קבוצה ריקה, כלומר עדיין בוטום. הזוג <⊥,⊤> נכנס אל הבלוק של שורה 3 בה הוא לא משתנה, ומפעפע לשתי הקשתות שמחוברות אליה: שורה 4 ושורה 9.

נבצע לפי הסמנטיקה שלנו את החישוב בשורה 6:

$$[\![x.replace('s','t')]\!]^{\#}(<\top,\bot>) = \Sigma \setminus \{'s'\} \cup \{'t'\} = \Sigma \setminus \{'s'\}$$

ונבצע את ההשמה של הערך הזה ל-*y*.

באופן דומה, נבצע את החישוב בשורה 8:

$$[\![(x.upper(\quad)+'w') * 3]\!]^{\#}(<\top,\bot>) = [\![x.upper(\quad)+'w']\!]^{\#}(<\top,\bot>)$$
$$= [\![x.upper(\quad)]\!]^{\#}(<\top,\bot>) \cup \{'w'\} = \{upper(c) | c \in \Sigma\} \cup \{'w'\}$$
$$= \{'A'-'Z','0'-'9','w'\}$$

ונבצע את ההשמה אל *y*.

כעת נחשב מחדש את הכניסה אל שורה 3, שכן זהו *join* של שלושה ענפים: היציאה משורה 2, היציאה משורה 6, והיציאה משורה 8:

$$< \top \cup \top \cup \top, \bot \cup \Sigma \setminus \{'s'\} \cup \{'A'-'Z','0'-'9','w'\} >=< \top, \Sigma \setminus \{'s'\} >$$

ושוב נעביר ערך זה לשתי היציאות של שורה 3. למזלנו החישובים של שורה 6 ו-8 עבור $\sigma^{\#}$ החדשה יוצאים זהים (שכן הם לא נסמכים על y ו-x לא השתנה) וכאן האנליזה מתכנסת.

מה שיש לנו כעת אלה ערכי ה"זיכרון" של התכנית בכל נקודה. מה שאין לנו זו יכולת לקבוע משהו לגבי ה-*assert*.

ג. (5 נק') הגדירו את הסמנטיקה האבסטרקטית של find ובדקו את ה-assert-ים בתכנית.

1. מכיוון שבסמנטיקה הקונקרטית find מחזירה מספר ולא מחרוזת, כמובן שלא ניתן להשתמש בדומיין שהגדרתם בסעיף א' לערך ההחזרה שלה. לפיכך הגדירו דומיין חדש שמיועד לערך ההחזרה של find ולבדיקת assert-ים המכילים אותו. תארו את האיברים, את יחס הסדר, ואת האבסטרקציה. כמו כן התייחסו לשתי פעולות בסמנטיקה האבסטרקטית: == ו-=!.

<u>הנחה מקלה:</u> השימוש היחיד ב-find יהיה בתוך ביטוי בוליאני הבודק האם מחרוזת נמצאת או לא נמצאת בתוך מחרוזת שניה, כמו ב-assert-ים בשורות 9 ו-10.

לפי ההנחה המקלה, הערכים היחידים של find שמעניינים אותנו הם 1-, לא 1-, ולא ידוע. ניתן להשתמש פה באחד הדומיינים המספריים שלמדנו בכיתה (אך שימו לב כי אם השתמשתם בדומיין הסימנים, להשתמש בגרסה שלו שבה $0 \sqsubseteq -$ תוביל אתכם למצב שבו אתם כבר לא יכולים לבדוק את האסרט בוודאות)

ניתן גם להגדיר דומיין חדש וייעודי לנושא. איבריו יהיו: $\{-1, \overline{-1}, \bot, \top\}$, כלומר שלושת המצבים שצוינו לעיל ותוספת "לא מוגדר". בין -1 ו-$\overline{-1}$ אין יחס סדר, כל האיברים קטנים מ-$\top$ וכל האיברים גדולים מ-$\bot$. פעולת join נובעת מידית מיחס הסדר.

נגדיר גם את האבסטרקציה והקונקרטיזציה כך:

$$\beta(x) = \begin{cases} -1 & x = -1 \\ \overline{-1} & x \neq -1 \end{cases}$$
$$\alpha(X) = \sqcup \{\beta(x) \mid x \in X\}$$
$$\gamma(\overline{-1}) = \mathbb{Z} \setminus \{-1\}$$
$$\gamma(-1) = \{-1\}$$
$$\gamma(\top) = \mathbb{Z}$$
$$\gamma(\bot) = \emptyset$$

כעת נותר רק להגדיר את הסמנטיקה האבסטרקטית לשתי הפעולות שמעניינות אותנו:

הגרסה האבסטרקטית של == תחזיר אמת אם היא משווה בין -1 ל--1 או בין $\overline{-1}$ ל-$\overline{-1}$. היא תחזיר שקר רק אם היא משווה בין $\overline{-1}$ ל--1. בכל מקרה אחר, לא ניתן להכריע. באופן דומה, עבור !=, השוואה בין -1 ל--1 או בין $\overline{-1}$ ל-$\overline{-1}$ תחזיר שקר, בין $\overline{-1}$ ל--1 תחזיר אמת, ובכל שאר המקרים לא ניתן להכריע. בקיצור, ניתן להכריע רק את המצבים בהם החיתוך בין המצבים הקונקרטיים ריק.

2. השתמשו בדומיין החדש כדי להגדיר את הסמנטיקה האבסטרקטית של `find`. כמו בסעיף א', ניתן להניח תוצאת ריצה של constant propagation וניתן לפצל למקרים.

נגדיר את find כך:

$$[\![x.find(y)]\!]^{\#}\sigma^{\#} = \begin{cases} -1 & y \text{ is constant}, len(y) = 1, [\![y]\!]^{\#}\sigma^{\#} \cap [\![x]\!]^{\#}\sigma^{\#} = \emptyset \\ \top & \text{אחרת} \end{cases}$$

כלומר: במידה וידוע כי y הוא תו בודד, אז אם הוא לא נמצא בקב' המייצגת את x נחזיר *1-* שכן אנחנו יודעים בודאות שכך *find* הקונקרטי יתנהג. בכל מקרה אחר, אנחנו לא מסוגלים לקבוע מה יחזיר *find* ולכן נחזיר $\top$.

חדי אבחנה ישימו לב כי בעצם $\overline{-1}$ הוא איבר חסר תועלת בדומיין שלנו כי *find* לעולם לא יחזיר אותו, ומובטח לנו כי לעולם לא נידרש לו בצד השני של == או !=. לכן ניתן להגדיר את הדומיין בלעדיו, עם האיברים $\{-1, \bot, \top\}$ בלבד.

3. בדקו את ה- assert בשורה 9. האם הוא מתקיים?

כעת ניתן לחשב את האסרט לפי הסמנטיקה של find, שבלעדיה לא יכולנו לומר דבר ולא ניתן היה לראות שום דבר על תוצאת ה-assert:

$$[\![y.find('s') == -1]\!]^{\#}(< \top, \Sigma \setminus \{'s'\} >) =$$
$$[\![y.find('s')]\!]^{\#}(< \top, \Sigma \setminus \{'s'\} >) == -1$$

שימו לב כי אלו 1- האיבר האבסטרקטי ופעולת == שהגדרנו בתת-סעיף 1

כעת, מכיוון ש-y קבוע באורך 1, ו-$(\Sigma \setminus \{'s'\}) \cap \{'s'\} = \emptyset$, לפי הסמנטיקה של *find*:

$$-1 == -1$$

4. בדקו את ה-assert בשורה 10. האם הוא מתקיים? האם עבור הדומיין שהגדרתם בסעיף א' תיתכן תכנית שבה הוא מתקיים? נמקו.

עבור ה-assert בשורה 10, בעוד ש-'w' קבוע באורך 1, $(\Sigma \setminus \{'s'\}) \cap \{'w'\} \neq \emptyset$ ולכן לפי הסמנטיקה של *find*, נקבל T, ובעצם ה-*assert* יהיה על הבדיקה $T! = -1$ שאי אפשר להכריע את ערכה.
מעבר לכך, כפי שראינו בסעיף 2 כשהגדרנו את הסמנטיקה של find, תוצאת האנליזה פשוט לא מכילה את היכולת לבדוק האם תו כן מופיע – מכיוון שהאיברים האבסטרקטיים מייצגים תווים שעשויים להופיע, אבל הקונקרטיזציה שלהם מכילה גם מחרוזות בהם התו לא מופיע, find לעולם לא תוכל להחזיר כי משהו באמת נמצא. לכן אף assert מהפורמט הזה לא יוכל להיות מוכרע עם אנליזה שיכולה להכריע את ה-assert בשורה 9.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<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:

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

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:

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

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:G=(V,E).

הצורה הכללית של המשוואות בחישוב סריקה קדמית:

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

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

הצורה הכללית של המשוואות בחישוב סריקה אחורית:

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

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

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
AUTO	auto
AND	and
OR	or
NOT	not
TRUE	true
FALSE	false
RETURN	return
IF	if
ELSE	else
WHILE	while
BREAK	break
CONTINUE	continue
SC	;
COMMA	,
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. $Program \rightarrow Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
5. $RetType \rightarrow Type$
6. $RetType \rightarrow VOID$
7. $Formals \rightarrow \epsilon$
8. $Formals \rightarrow FormalsList$
9. $FormalsList \rightarrow FormalDecl$
10. $FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
11. $FormalDecl \rightarrow Type\ ID$
12. $Statements \rightarrow Statement$
13. $Statements \rightarrow Statements\ Statement$
14. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
15. $Statement \rightarrow Type\ ID\ SC$
16. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
17. $Statement \rightarrow AUTO\ ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
19. $Statement \rightarrow Call\ SC$
20. $Statement \rightarrow RETURN\ SC$
21. $Statement \rightarrow RETURN\ Exp\ SC$
22. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
23. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
24. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
25. $Statement \rightarrow BREAK\ SC$
26. $Statement \rightarrow CONTINUE\ SC$
27. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
28. $Call \rightarrow ID\ LPAREN\ RPAREN$
29. $ExpList \rightarrow Exp$
30. $ExpList \rightarrow Exp\ COMMA\ ExpList$
31. $Type \rightarrow INT$
32. $Type \rightarrow BYTE$
33. $Type \rightarrow BOOL$
34. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
35. $Exp \rightarrow Exp\ BINOP\ Exp$
36. $Exp \rightarrow ID$
37. $Exp \rightarrow Call$
38. $Exp \rightarrow NUM$
39. $Exp \rightarrow NUM\ B$
40. $Exp \rightarrow STRING$
41. $Exp \rightarrow TRUE$
42. $Exp \rightarrow FALSE$
43. $Exp \rightarrow NOT\ Exp$
44. $Exp \rightarrow Exp\ AND\ Exp$
45. $Exp \rightarrow Exp\ OR\ Exp$
46. $Exp \rightarrow Exp\ RELOP\ Exp$
47. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$