

מבחן סוף סמסטר – מועד ב' טור מקור

מרצה אחראי: ד"ר הילה פלג

מתרגלים: הילה לוי, אלעד רון, תומר כהן, גיא ארבל

הוראות:

1. בטופס המבחן X 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 התשובות הנפרד בלבד. את התשובות לשאלות הפתוחות יש לכתוב במחברת הבחינה.
8. ודאו כי אתם מגישים טופס תשובות ומחברת בחינה בלבד.

בהצלחה!

חלק א' - שאלות סגורות (50 נק')

שלבי קומפילציה (20 נק')

נתונה התוכנית הבאה בשפת FanC:

```
1. bool is_palindrome (int a)
2. {
3.     int tmp = a;
4.     int reserved = 0;
5.     while (tmp > 0) {
6.         int digit = tmp - (tmp / 10) * 10; // Last digit
7.         reserved = reserved * 10 + digit;
8.         tmp = tmp / 10;
9.     }
10.    return reserved == a;
11. }
```

Handwritten notes:
- Line 1: ~~bool~~ → semantic error
- Line 4: reserved == 0; → syntactic error
- Line 9: tmp > 0; → syntactic error
- Line 10: return reserved == a; → lex

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

שאלה מספר 1:

(2 נק') מחליפים את האופרטור = לאופרטור == בשורה 3.

א. שגיאה בניתוח תחבירי

- ב. שגיאה בניתוח לקסיקלי
- ג. שגיאה בניתוח הסמנטי
- ד. אין שגיאה
- ה. שגיאה בייצור קוד
- ו. שגיאה בזמן ריצה

שאלה מספר 2:

(2 נק') מחליפים את שורה 8 ל- tmp /= 10;

א. שגיאה בניתוח תחבירי

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

שאלה מספר 3:

(2 נק') משנים את שורה 4 ל- `int reserved = 0b;`

א. אין שגיאה

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

שאלה מספר 4:

(2 נק') משנים את חתימת הפונקציה `is_palindrome` כך שתחזיר `int`.

א. שגיאה בניתוח סמנטי

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

שאלה מספר 5:

(2 נק') הופכים את הסוגריים המסולסלים { } בשורות 5 ו-9 לסוגריים מרובעים [].

א. שגיאה בניתוח לקסיקלי

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

שינויים ב-FanC- שאלות 6-7:

נרצה להוסיף את המילה השמורה rand לשפת FanC. מילה זו תייצג מספר רנדומלי (int) שהוגרל בתחילת הקומפילציה ולאחר מכן יישאר קבוע לאורך כל הריצות של התוכנית עד לפעם הבאה בה יקמפלו את התוכנית. עבור כל rand בתוכנית נקבל מספר רנדומלי אחר. לדוגמה עבור:

```
int a = rand;  
int b = rand;
```

נקבל את הקוד llvm:

```
t0 = add i32 0, 22  
t1 = add i32 0, 90
```

כאשר המספר הראשון שהקומפיילר הגריל הוא 22 והמספר השני הוא 90.

שאלה מספר 6:

(5 נק') באיזה שלב הכי מוקדם של הקומפיילר ניתן להגריל את המספר?

א. ניתוח לקסיקלי

ב. ייצור קוד

ג. ניתוח תחבירי

ד. ניתוח סמנטי

ה. זמן ריצה

שאלה מספר 7:

(5 נק') הוחלט להגריל את המספר בשלב ייצור הקוד, מה הם שלבי הקומפילציה שנהיה חייבים לשנות מלבד שלב ייצור קוד?
semantic

א. ניתוח לקסיקלי, ניתוח תחבירי, ניתוח סמנטי

ב. ניתוח לקסיקלי

ג. ניתוח לקסיקלי, ניתוח תחבירי

ד. ניתוח תחבירי, ניתוח סמנטי

ה. אף תשובה אינה נכונה

אופטימיזציות (10 נק')

נתון הקוד הבא בשפת הביניים, נוסף את return לפעולות בשפת הרביעיות. המשתנה n מתקבל כקלט בתחילת הפונקציה, המשתנים t ו-t2 הם משתנים זמניים שלא חיים מחוץ לבלוק הבסיסי. כל שאר המשתנים הם משתני הפונקציה.

```
1. a = 0
2. b = 1
3. if (n = 0) goto 11
4. a = n
5. t = a + b
6. a = b
7. b = t
8. t2 = 4 + 1
9. n = n - t2
10. goto 3
11. return a
```

1. Dead Code Elim
2. Const Prop
3. Const Fold
4. AS

שאלה מספר 8:

(5 נק') כמה leaders מזהים על ידי אלגוריתם יצירת ה-CFG על הקוד הנתון?

- א. 4
- ב. 2
- ג. 3
- ד. 5

שאלה מספר 9:

5 נק') בהנחה שלא נתונה אף תוצאה של אנליזה על הפונקציה, הריצו על הבלוק שמכיל את שורה 6 את האופטימיזציות הבאות : Algebraic Simplification, Constant Propagation, Constant Folding, dead code elimination מספר רב של פעמים ככל הניתן.
מהי התוצאה אליה הגעתם?

```
a = n
t = a + b
a = b
b = t
n = n - 5
goto 3
```

א.

```
t = n + b
b = t
n = n - 5
goto 3
```

ב.

```
a = n
t = a + b
b = t
n = n - 5
goto 3
```

ג.

```
t = n + b
a = b
b = t
n = n - 5
goto 3
```

ד.

```
t = a + b
a = b
b = t
n = n - 5
goto 3
```

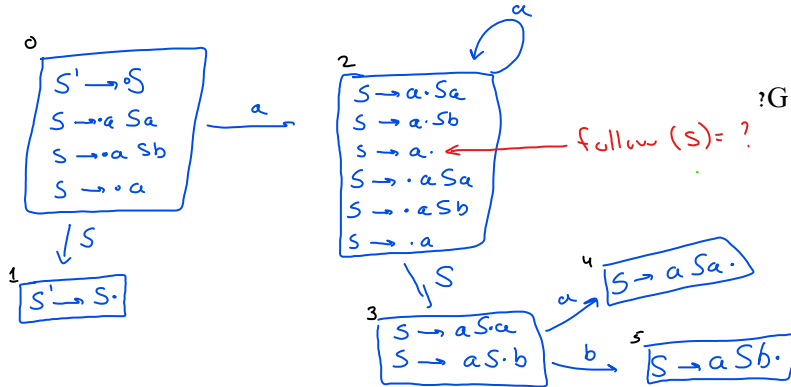
ה.

דקדוקים (20 נק'):

- 0 $S' \rightarrow S$
- 1 $S \rightarrow a S a$
- 2 $S \rightarrow a S b$
- 3 $S \rightarrow a$

$$\text{follow}(S) = \{a, b\}$$

יהי G הדקדוק הבא:



שאלה מספר 10:

(4 נק') מהו מספר המצבים באוטומט LR(0) של G?

	a	b	#	S
0	s2 s2			1
1				
2	s2/r3 s2/r3	r3 r3		3
3	s4 s4	s5 s5		
4	r4 r4	r4 r4		
5	r2 r2	r2 r2		

LR(0)
SLR

- א. 6
- ב. 5
- ג. 4
- ד. 7
- ה. 8

שאלה מספר 11:

(4 נק') מהו השינוי במספר התאים בעלי קונפליקט בטבלת הניתוח SLR של הדקדוק לעומת טבלת הניתוח LR(0) של הדקדוק G?

א. נשאר זהה

- ב. קטן בלפחות 2
- ג. גדל ב-1
- ד. קטן ב-1
- ה. גדל בלפחות 2

שאלה מספר 12:

(4 נק') בכמה תאים בטבלת הניתוח SLR של הדקדוק G קיים קונפליקט אחד לפחות?

א. תא אחד

- ב. אין תאים כאלה
- ג. 2 תאים
- ד. 3 תאים
- ה. יותר מ-3 תאים

שאלה מספר 13:

(4 נק') לכל שני דקדוקים $G1 = (V, T, P, S)$ ו- $G2 = (V', T', P', S')$ עבורם $S \neq S'$ נגדיר את הדקדוק הבא: $G3 = (V \cup V' \cup \{S''\}, T \cup T', P \cup P' \cup \{S'' \rightarrow S, S'' \rightarrow S'\}, S'')$ סמנו את הטענה הנכונה ביותר:

א. אף אחת מהתשובות האחרות אינה נכונה.

- ב. $G3$ ב-LL(1) לכל $G1$ ו- $G2$.
- ג. $G3$ לא ב-LL(1) לכל $G1$ ו- $G2$.
- ד. $G3$ ב-LL(1) אם $G1$ ו- $G2$ ב-LL(1).
- ה. $G3$ ב-LL(1) אם ורק אם $G1$ ו- $G2$ ב-LL(1).

(4 נק') נתון הדקדוק הבא :

יוסי פיתח את המנתח LR(1) של הדקדוק (הדקדוק נמצא במחלקה, אין צורך לוודא זאת) בחר מילה כלשהי שנשמנה w אשר נמצאת בודאות בשפה!

יוסי הריץ את אלגוריתם המנתח.

נסמן :

$$a^* b c^*$$

$t = 0$ הזמן בעת המצב ההתחלתי

aabcc

א. קיים t עבורו מתקיים $a(t) < c(t)$

$$[a] \rightarrow \begin{bmatrix} a \\ a \end{bmatrix} \rightarrow \begin{bmatrix} b \\ a \\ a \end{bmatrix} \rightarrow \begin{bmatrix} s \\ a \\ a \end{bmatrix} \rightarrow \begin{bmatrix} s \\ a \end{bmatrix} \rightarrow \begin{bmatrix} s \\ a \end{bmatrix} \rightarrow [s] \rightarrow [s'] \rightarrow acc$$

ד. $c(t) \leq S(t)$ לכל t מתקיים ✓

ה. $c(t) \leq a(t)$ לכל t מתקיים ✓

חלק ב' - שאלות פתוחות (50 נק')

שאלה 1: ייצור קוד (20 נק')

הוסיפו לשפת FanC מבנה בקרה חדש, `if_according_to_b_2`:

הדקדוק שאיתו ממומש `if_according_to_b_2` הוא:

$$\begin{aligned} S &\rightarrow \text{if_according_to_b}_2(E; B_list) \{S_1\} \\ B_list &\rightarrow B_list_1, B \\ B_list &\rightarrow B \end{aligned}$$

משמעות המבנה: נסמן את הערך של E ב- k . אם k שלילי או גדול ממספר הביטויים הבוליאניים ב- B_list אין לבצע דבר. אחרת, נפעל באופן הבא: אם ישנם k ומעלה ביטויים בוליאניים שמשוערכים ל-true אז יש לבצע את S_1 ואחרת אין לבצע אותו.

דוגמה למבנה הבקרה החדש:

```
if_according_to_b_2 (3 ; x > 2 , 3 > 2 , 4 > 3 , 4 > 5)
{
    printf("Good Luck!");
}
```

עבור $x = 3$ הערך שיודפס הוא: Good Luck!
עבור $x = 1$ לא יודפס דבר.

שימו לב:

- ניתן להניח שבקוד לא יהיו שגיאות קומפילציה
- אין לשנות את הדקדוק פרט להוספת מרקרים M, N
- ניתן להשתמש במרקרים M, N שנלמדו בכיתה בלבד
- למשתנים S, E, B ישנן התכונות שהוגדרו בכיתה בלבד
- למשתנים S, E, B ישנם כללי גזירה פרט לאלה המוצגים בשאלה
- אסור להשתמש במשתנים גלובליים
- המשתנה S תמיד יכול `nextlist`

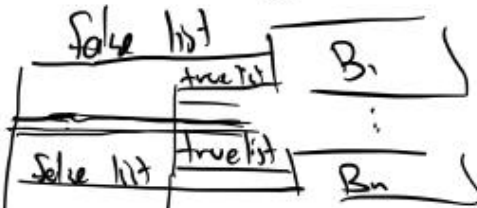
1. הציעו פריסת קוד המתאימה לשיטת `backpatching` עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות הסמנטיות שאתם משתמשים בהן עבור כל משתנה.
2. כתבו סכימת תרגום בשיטת `backpatching` המייצרת את פריסת הקוד שהצעתם בסעיף הקודם. על הסכימה להיות יעילה ככל האפשר, הן מבחינת זמן הריצה שלה והן מבחינת המקום בזיכרון שנדרש עבור התכונות הסמנטיות.

S

$E \rightarrow t$

goto init

①



s_1 next list

next list

init:

② if $t > n$ goto _____
if $t < 0$ goto _____

$i = 0;$
 $count = -1;$

→ true: $count++;$
③ if $count \geq t$ goto s_1

→ false $i++;$

④ if $i > n$ goto _____

if $i == 1$ goto B_1

⑤ \vdots

if $i == n$ goto B_n

- B-LIST

true list

false list

quest list

B-LIST \rightarrow M B

{ $\begin{matrix} \text{הכניסה} & \text{ל} & \text{שליש} \\ \text{fb} & , & \text{tb} \end{matrix}$ }

B-LIST \rightarrow B-LIST, M B

{ $\begin{matrix} \text{הכניסה} & \text{ל} & \text{שליש} \\ \text{B} & \text{ו} & \text{fb} , \text{tb} \\ & & \text{הכניסה} \end{matrix}$ }

S \rightarrow if--b₂ (E N ; B-LIST) {M S₁}

① BR (N.root list, Next queue);

② { $\begin{matrix} \text{שם} & \text{ל} & \text{הכניסה} & \text{ל} & \text{שליש} \\ \text{S.next list} & & & & \text{הקנייה} \end{matrix}$ }

BR (B-LIST.true list, Next queue)

③ emit (if count > B-list.size goto M.queue)

BR (B-LIST.false list, Next queue)

{ S.next list } δ $\begin{matrix} \text{שם} & \text{ל} & \text{הכניסה} & \text{ל} & \text{שליש} \\ \text{הקנייה} & & & & \end{matrix}$ ④

for j in M

⑤ emit (if i == j goto B-list.queue[j])

שאלה 2: אנליזה סטטית (30 נק')

```
//question2.h

int foo(double a1) /*foo_1*/ {...}
std::string foo(float a1) /*foo_2*/ {...}

int bar(int a1, int a2) /*bar_1*/ {...}
float bar(int a1, float a2) /*bar_2*/ {...}
std::string bar(std::string a1, int a2) /*bar_3*/ {...}

std::wstring qux(float a1) /*qux_1*/ {...} //(wstring is a Unicode string)
std::string qux(std::string a1) /*qux_2*/ {...}

int thud() /*thud_1*/ {...};
double thud() /*thud_2*/ {...};
float thud() /*thud_3*/ {...};

bool randbool() { /*returns a random boolean*/ }
```

מייקל אחראי על קומפיילר C++ שכונתי ורוצה לשלב באופן מתקדם בין שני פיצ'רים של השפה: העמסת פונקציות ו-`auto`.

מייקל רוצה לחלק את הבדיקות הסמנטיות לשני שלבים: בשלב הראשון ייקבעו הטיפוסים של כל הביטויים שלא מעורבת בהם פונקציה מועמסת (למשל בשלב זה ייקבע כי טיפוס הביטוי `(bool randbool())`, יירשם בטבלת הסמלים טיפוס כל המשתנים שהטיפוס שלהם מופיע בהגדרה, וכן יירשם טיפוס משתנים המוגדרים עם `auto` אבל כבר נקבע טיפוס הביטוי שמושם אליהם (כלומר: אין בו שימוש בפונקציות מועמסות או משתנים אחרים שעוד לא נקבע הטיפוס שלהם). ההחלטה לגבי איזו גרסה של פונקציה מועמסת תיקרא ומה יהיה הטיפוס של משתנה `auto` המקבל השמה התלויה בפונקציה מועמסת תידחה לשלב שני חדש אחרי השלב הסמנטי אך לפני ייצור הקוד.

מייקל רוצה שגם אם לכאורה ייתכנו מספר אפשרויות טיפוס בקריאה ספציפית, כל עוד לכל השמה יש רק אפשרות יחידה לטיפוס שנכונה לכל הריצות האפשריות, וכל ההשמות לאותו משתנה תואמות בכל רחבי התכנית, הבדיקה החדשה תכריע מה הטיפוס של המשתנים וטבלת הסמלים תאפשר להשתמש בטיפוס החזרה כאינפורמציה נוספת על מנת להכריע בין העמסות.

כך למשל בתכנית הקצרה לדוגמה:

```
auto a = bar(0,2.0); //definitely bar_2
std::string b = foo(a);
```

קביעת הטיפוס של `a` וכן ההעמסה הספציפית של `foo` ו-`bar` יידחו לשלב החדש. בשלב החדש, ההעמסה של `bar` נקבעת באופן יחיד לפי טיפוס הפרמטרים, ולכן הטיפוס של `a` ייקבע כ-`float`. שתי הקריאות של `foo` יכולות לקבל את `a` כפרמטר, אבל מכיוון שהטיפוס של `b` כבר נקבע כ-`string`, ניתן להשתמש במידע הנוסף כדי להכריע ביניהן. אם לא ניתן היה להכריע ביניהן, התכנית לא הייתה מתקמפלת.

שימו לב כי משמעות הדבר שייתכן שתוצאת השלב החדש היא שהתכנית לא מתקמפלת, אבל במידה והיא כן, בשלב ייצור הקוד של הקומפיילר כבר יהיה ידוע איזו פונקציה תיקרא. מייקל מטיל על גייסון ואלינור לפתח את הפיצ'ר. גייסון מבולבל.

א. (3 נק')

1. הסבירו לגייסון מדוע פונקציה מועמסת (overloaded) בניגוד לפונקציות נדרסות בירושה (polymorphism) נבחרת לפני שלב ייצור הקוד.
פונקציות מועמסות שונות זו מזו בטיפוס הפרמטרים, אז בזמן הבדיקה הסמנטית אפשר לדעת לפי טיפוס הארגומנטים איזו פונקציה לבחור כל עוד אין דו משמעות. לדריסות שונות של פונקציה על טיפוסים בהיררכיית ירושה יש כולן אותה חתימה! כלומר אי אפשר להבדיל ביניהן כך, וחייבים לחכות לזמן ריצה ולבחור אותן לפי הטיפוס הדינמי של האובייקט.
2. אלינור אומרת שבניגוד למימוש המקורי של auto, אי אפשר יהיה לבצע את ההחלטה לגבי הטיפוס הסופי של משתנה שמוגדר auto ביחד עם בדיקות הטיפוסים של הקומפילר, ויהיה צורך באנליזה מתקדמת יותר. הסבירו מדוע אלינור צודקת.
בגלל שיש תלות הדדית בין משתני auto לפונקציות מועמסות, כל עוד לא בטוחים איזו פונקציה צריכה להיבחר יכול להיווצר מצב בו ל-auto יש כמה טיפוסים אפשריים. למשל:

```
auto x = thud();
```

ל-x יש שלושה טיפוסים אפשריים.

ואז אם משתמשים ב-x כדי לקרוא לפונקציה מועמסת, הכמה טיפוסים מתורגמים לכמה העמסת אפשריות. אבל יכול להיות שאפשר להגיע להכרעה מהו הטיפוס היחיד שקונסיסטנטי (כל עוד בכל נקודה בקוד הוא יחיד), ולכן אנליזה תסדר אותנו.

ב. (4 נק') אלינור מבקשת מצידי לעזור לה לממש את האנליזה. הסבירו לאלינור ולצידי בקצרה מה צריך לדעת על מנת לבחור בין מספר העמסות אפשריות של אותה הפונקציה, מה תצטרך אנליזה סטטית לבדוק על מנת לעשות זאת, וכיצד ישתמשו בתוצאות שלה.
למרות שיש פה שני דברים עם ריבוי אפשריות, שימו לב שטיפוס המשתנים קובע את טיפוס הפונקציות, אז אפשר לעקוב אחריהם בלבד, כשההפך אינו נכון. לכן נעקוב אחרי הטיפוסים של ביטויים ומשתנים – במצב האבסטרקטי שלנו נשמור את המשתנים – ונחפש טיפוס שמופיע בכל המסלולים. כלומר: כאשר מאחדים מסלולים, נרצה רק את מה שמגיע משני הצדדים. אם הוא יחיד וקונסיסטנטי בכל הנקודות בקוד, מצאנו את הטיפוס שלנו.

ג. (7 נק') עזרו לאלינור וצידי להגדיר את האנליזה:

1. הגדירו את הדומיין: מהם האיברים? מהו יחס הסדר \sqsubseteq ? מהי פעולת ה-join? \sqcup ?
אם קבוצת כל הטיפוסים היא T, ניקח את דומיין החזקה של T, כלומר איברים יהיו קבוצות של טיפוסים. מכיוון שאנחנו רוצים את מה שמגיע משתי הכניסות של כל איחוד, נבחר את יחס הסדר להיות \supseteq ואת ה-join להיות \cap . הרבה תשובות בחרו את יחס הסדר להיות הפוך, וגילו אחרי שהריצו את האנליזה בסעיף ד שהיא בעצם רק צברה עוד ועוד אפשרויות לטיפוסים ולכן אי אפשר היה להכריע כלום.
כדי לעקוב אחרי משתני תכנית נגדיר את המצב האבסטרקטי $\sigma^\#$ להיות פונקציה משם משתנה לאיבר בדומיין החזקה של T. דומיין המכפלה $P(T)^k$ כאשר יש k משתנים זה שקול, סתם יותר קשה לתחזוקה.

2. הגדירו את פונקציות האבסטרקציה α ופונקציית הקונקרטיזציה γ המקשרות בין הערכים האבסטרקטיים והקונקרטיים. (רמז: מומלץ להגדיר אותן עבור משתנה בודד.)

הדבר החשוב פה הוא שגם הדומיין האבסטרקטי וגם הדומיין הקונקרטי הם דומיינים של קבוצות טיפוסים אפשריים עבור ביטויים/משתנה. זה בעצם אומר שפונקציית β שמתרגמת איבר קונקרטי יחיד לאיבר אבסטרקטי יחיד משאירה את בדיוק אותם איברים.

מפה, $\alpha(S)$ על S קבוצת איברים קונקרטיים היא join של הפעלת β על כל אחד מהם, או במילים אחרות הטיפוסים שמופיעים בכל $c \in S$.

$\gamma(a)$ על איבר אבסטרקטי a יהיה כל אופציות הטיפוסים הקונקרטיים ש-a מתאר, כלומר כל קבוצות הטיפוסים שמכילות לפחות את כל הטיפוסים ב-a.

3. הגדירו את הסמנטיקה עבור ביטוי קריאה לפונקציה ועבור משפט השמה.

הניחו כי נתונה לכם טבלת הסמלים symtab שהיא תוצאה של השלב הסמנטי הראשון של הקומפילר, וניתן לתשאל לפי שם וטיפוס פרמטרים ולקבל את החתימות של כל הפונקציות שיתאימו לקריאה כזו. למשל,
 $\text{symtab}(\text{randbool}())$ יחזיר קבוצה בגודל אחד, $\{() \rightarrow \text{bool}\}$, ו- $\text{symtab}(\text{bar}(\text{int}, \text{char}))$ יחזיר את הקבוצה $\{(int, int) \rightarrow int, (int, float) \rightarrow float\}$ מכיוון שערך מטיפוס char ניתן להשמה גם ל-int וגם ל-float.

משפט השמה: $\llbracket x = e \rrbracket^{\sigma^\#} = \sigma^\#[x \mapsto \llbracket e \rrbracket^{\sigma^\#}]$.

חלק מהתשובות התפתו להוסיף כבר כאן \sqcup עם הערך הקודם של המשתנה. זו דרך להכריח את האנליזה להתכנס יותר חזק, אבל עשויה לפספס מצבים שבהם השמה כלשהי מייצרת דו משמעות שנרצה להודיע עליה במקום להתכנס בכוח, למשל אם הייתה עוד השמה לפני שורה 3 של analyze_me.

ביטוי קריאה לפונקציה:

$$\llbracket f(a_1, \dots, a_k) \rrbracket^{\sigma\#} = \bigcup \{ \text{symtab}(foo(\tau_1, \dots, \tau_k)) \mid \tau_1 \in \llbracket a_1 \rrbracket^{\sigma\#}, \dots, \tau_k \in \llbracket a_k \rrbracket^{\sigma\#} \}$$

דגשים: 1. לא ביקשנו מכם לטפל בשאר הביטויים אז אפשר להניח שהם יטופלו לבד. 2. חשוב לשים לב שלארגומנטים יכולות להיות מספר אופציות וצריך לבדוק את כולן. 3. מדובר בביטוי ולכן הוא מחזיר ערך בדומיין ולא מעדכן את המצב האבסטרקטי.

```

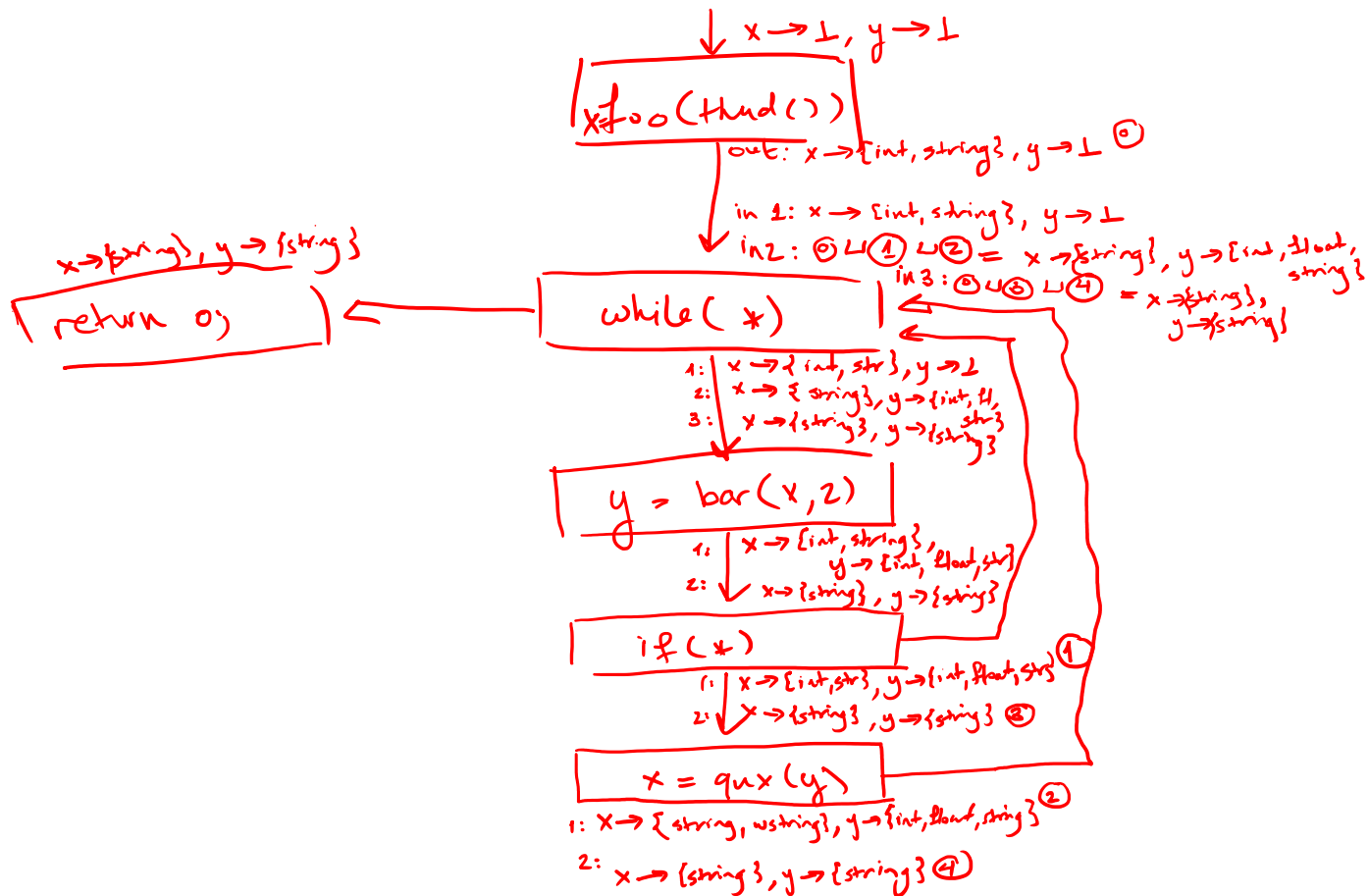
1. int analyze_me()
2. {
3.     auto x = foo(thud());
4.     while (randbool()) {
5.         auto y = bar(x,2);
6.         if (randbool()) {
7.             x = qux(y);
8.         }
9.     }
10.    return 0;
11. }

```

ד. (10 נק') נתחו את הפונקציה analyze_me:

1. ציירו את ה-CFG של הפונקציה.
2. השתמשו באנליזה שלכם כדי לנתח את הפונקציה. יש להראות גם את ערכי הביניים של האנליזה לפני שהתייצבה. לנחותכם, להלן טבלה של type coercions מובנים בין הטיפוסים הבסיסיים ב-C++:

לטיפוס	ניתן לבצע implicit coercion מהטיפוסים
int	bool
double	bool,int,float
float	bool,int,double



ה. (4 נק') השתמשו בתוצאות האנליזה שלכם כדי לקמפל את הקוד : עבור כל קריאה לפונקציה מועמסת ציינו אחד מהשניים :

1. איזו מבין הפונקציות המועמסות תיבחר (לשם הנוחות, ניתנו להם שמות ייחודיים בהערה) ומדוע, או
2. הפרמטרים להדפסת הודעת השגיאה, כלומר איזה ניסיון קריאה מוביל לאיזו פונקציה או פונקציות.

למשל, אם האנליזה החליטה שעבור הקריאה ל-`qux` הפרמטר יכול להיות `int` או `string`, יש לציין :

`qux(int): qux_1`

`qux(string): qux_2`

שימו לב כי יש ארבע קריאות מועמסות שעליכם לקבוע.

את `thud` בשורה 3 לא ניתן להכריע : `thud_1`, `thud_2`, `thud_3` עדיין על השולחן בהיעדר פרמטרים :

`thud(): thud_1`

`thud(): thud_2`

`thud(): thud_3`

הפרמטר של `foo` בשורה 3 יכול להיות `int`, `double` או `float`, כל אחד מהם יכול להיות פרמטר גם ל-`foo_1` וגם ל-`foo_2` :

`foo(int): foo_1`

`foo(int): foo_2`

`foo(double): foo_1`

`foo(double): foo_2`

`foo(float): foo_1`

`foo(float): foo_2`

טעות נפוצה הייתה לקבוע את `x` להיות `string` ולכן להכריע גם את `foo` (ולפעמים גם את `thud`) מכאן, אבל חשוב לשים לב שכדי להכריע את `x` צריכה להיות לו אפשרות יחידה בכל השמה אליו בתכנית, ובשורה 3 עדיין יש שתי אפשרויות.

בשורה 5 הקריאה ל-`bar` תהיה `bar_3`, ובשורה 7 תיקרא `qux_2`. אין צורך להכריע את `randbool`, היא לא מועמסת.

1. (2 נק') טהאני מצטרפת לצוות ואומרת שהמרצה שלה לקומפילציה, גרייס הופר, המציאה פעם אנליזה—לא משנה מה הפרטים שלה—שתוכל לקבל חלק מההחלטות לפני האנליזה הנוכחית ותשפר את הביצועים של האנליזה של צידי ואליוור. אם האנליזה של טהאני בוחרת את הקריאה ל-thud בשורה 3 להיות קריאה ל-thud_1, האם צידי ואליוור יוכלו להשתמש במידע זה כדי לגרום לתכנית להתקמפל? אם כן, כיצד? אם לא, מדוע?

האנליזה לא תעזור: גם אם בחרנו בכוח את thud_1, היא מחזירה int, וקיימת המרה implicit ל-float ו-double, כלומר עדיין שתי הגרסאות של foo ייתכנו לפי טיפוס הפרמטרים ועדיין יש דו משמעות.