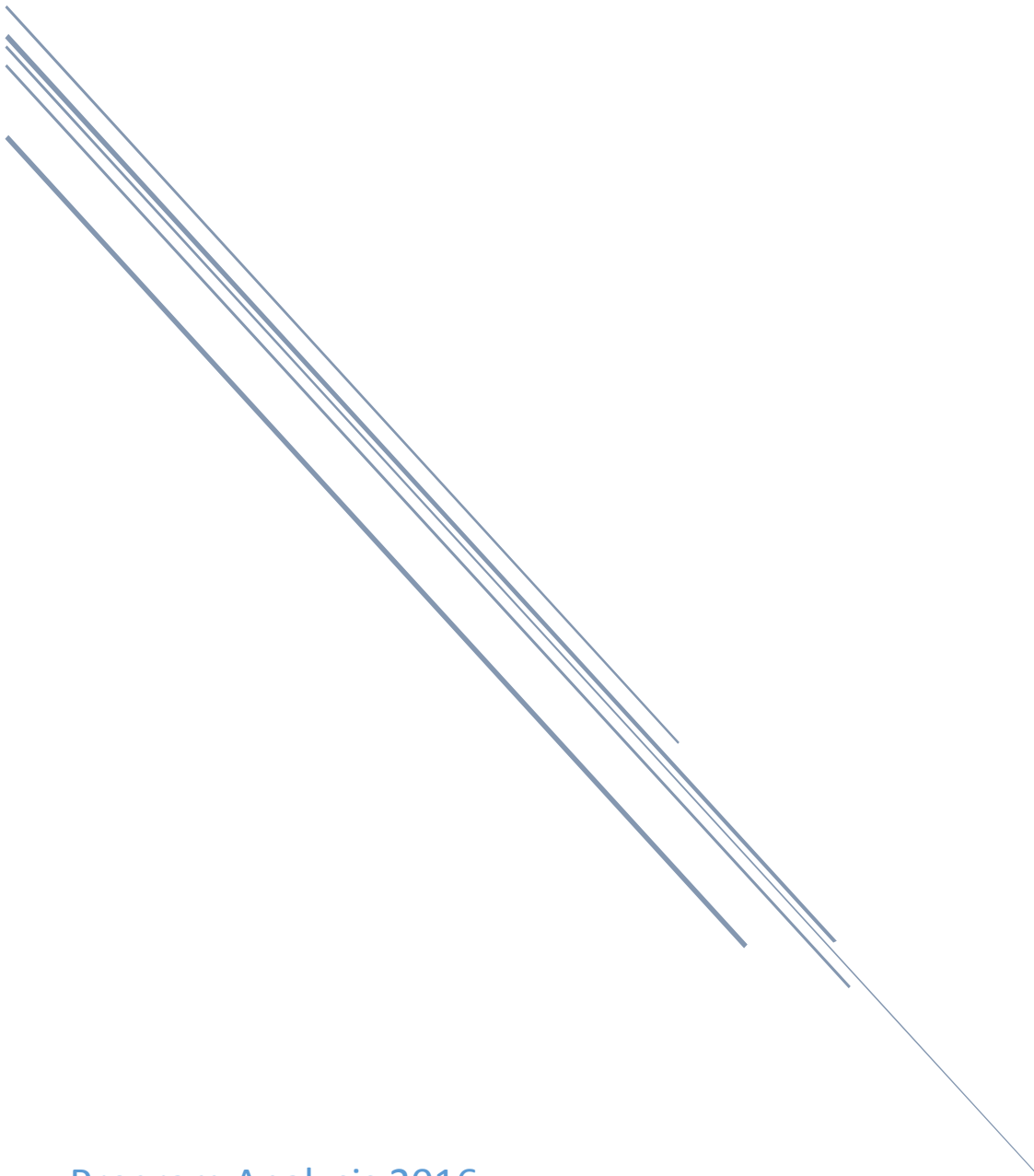


PYTHON SLICER (PROJECTOR)

Oded Elbaz, Tomer Greenwald



Program Analysis 2016

Lecturer: Prof. Mooly Sagiv

2.....	מבוא.....
4.....	איך להריץ?.....
5.....	כיצד לקרוא את הפלט.....
7.....	הנחות עבודה.....
8.....	אלגוריתם האנליזה.....
11.....	Points-to analysis.....
12.....	דוגמאות – Points-to analysis.....

בחרנו לממש Python Slicer עפ"י הצגת הפרויקט עבור Java בשיעור 12 ואישורך במייל לבצע את הפרויקט על קוד פיית'ון. קראנו לתוכנית Projector כיוון שהיא מקרינה את קוד הקלט על פני משתנה שבחר המשתמש.

Projector מבצע אנליזה על הקוד ויוצר שני גרפים בעלי רשימת קודקודים משותפת.

כל קודקוד ברשימה מתאים לשורת קוד בתוכנית המקורית. ישנם שני סוגי קודקודים:

- Statement Node: מתאים לשורת השמה או ביטוי.
- Control Node: מתאים לשורת קוד שמשפיעה על flow בתוכנית. לדוגמא if, else, while.

באשר לקשתות, סט אחד מתאר את Control Flow בתוכנית, והשני מתאר את התלות הלוגית (Dependency) בין שתי שורות קוד.

הכוח הגדול של האנליזה טמון בעובדה שהיא מזהה שינויים באובייקט עליו אנחנו מצביעים, גם כאשר השינויים בצעו ממצביע אחר.

תזכורת על שפת פיית'ון:

כל קריאה לConstructor יוצרת אובייקט חדש. שני משתנים יכולים להצביע אל אותו אובייקט. שינוי באובייקט משפיע על האובייקט ולא על המצביע, לכן נוכל לראות את השינוי ע"י גישה מכל אחד מהמשתנים שמצביעים אליו:

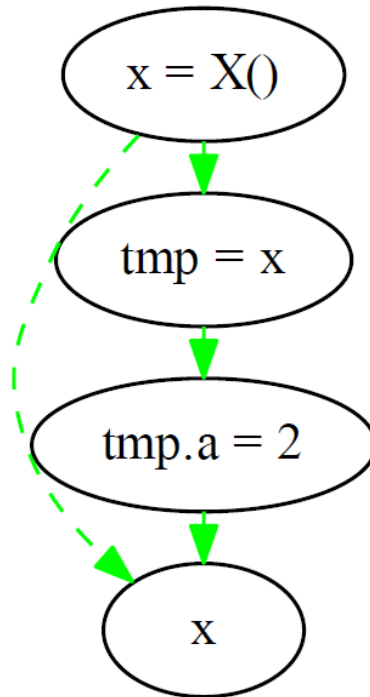
```
In [1]: class Example(object):
...:     pass
...:
In [2]: Example()
Out[2]: <__main__.Example at 0x30b8810>
In [3]: Example()
Out[3]: <__main__.Example at 0x30b89b0>
In [4]: e1 = Example()
In [5]: e2 = e1
In [6]: e1 is e2
Out[6]: True
In [7]: e1.a = 'Hello World'
In [8]: e2.a
Out[8]: 'Hello World'
```

האנליזה יודעת לזהות ששני משתנים מצביעים לאותו אובייקט, וכן שורות קוד שגורמות לשינוי באובייקט זה. אנחנו עושים זאת באמצעות Points-to Analysis כפי שיוסבר בהמשך. דוגמא (דוגמאות מורכבות יותר יינתנו בהמשך):

```
x = X()
tmp = x
tmp.a = 2
x
```

בדוגמא הזאת אנחנו מייצרים אובייקט חדש מסוג X שמצביעים אליו המשתנים x ו-tmp. אנחנו משנים את האובייקט דרך tmp, ורוצים לוודא שהמשתנה x מודע לשינויים הללו, כך שכאשר נקריין את הקוד על פני x, גם השורה tmp.a=2 תהיה בפלט (וכן השורה שמקשרת את tmp לאובייקט שעליו מצביע x).

זוהי תוצאת האנליזה:



אנחנו רואים ש- x בשורה האחרונה מושפע על ידי ההשמה הראשונה למשתנה וע"י ההשמה $tmp.a$.
מושפע מההשמה למשתנה tmp .

"מאחורי הקלעים" האנליזה מייצרת את הטבלה הבאה:

#line	assigned variable	influence variables	statement
0	x	$set([])$	$x = X()$
1	tmp	$set(['x'])$	$tmp = x$
2	$tmp\#a$	$set(['tmp'])$	$tmp.a = 2$
3		$set(['x', 'tmp\#a'])$	x

ניתן לראות שהאנליזה יודעת ש- x בשורה 3 מושפע מ- x ו- $tmp.a$.

למרות שהשורה $tmp = x$ משפיעה על x בשורה 3, tmp לא מופיע ברשימת המשתנים המשפיעים. בנינו את האנליזה כך כיוון שאנחנו לא רוצים להוסיף את כל המשתנים שמצביעים אל האובייקט, אלא רק אלו שמשפיעים על התוצאה. השורה $tmp = x$ תתווסף לפלט מכיוון ש- $tmp.a = 2$ תלוי ב- tmp . כך לדוגמא, אם נסתכל על הקוד הבא:

```

x = X()
tmp = x
tmp2 = x
tmp3 = x
tmp.a = 2
x
  
```

ההשמות של האובייקט אל $tmp2$ ו- $tmp3$ לא ישפיעו על x .

הנחת העבודה היא שהקוד בצורת SSA, כך שההשמה לכל משתנה היא יחידה. לכן, אם, לדוגמה, tmp.a נמצא ברשימת המשתנים המשפיעים, עלינו לחפש היכן בוצעה ההשמה היחידה לtmp.a.

כפי שניתן לראות כבר מהדוגמה הפשוטה הנ"ל, הקוד יודע לטפל בגישה לAttributes של משתנה. נשים לב שזוהי הדרך היחידה לשנות אובייקט. במילים אחרות, שינוי אובייקט הוא שינוי Attributes שלו.

לאחר שבנינו את הגרפים, הקרנת הקוד היא כבר עניין טכני – אנחנו הולכים מהסוף להתחלה עד אשר אנחנו מזהים את הפעם הראשונה שבה המשתנה המוקרן מושפע. משם אנחנו מטיילים אחורה על פני קשתות Dependency. כל פעם שאנחנו מגיעים לקודקוד חדש, אנחנו בודקים האם יש Control Flow בין שני הקודקודים. במידה ויש, אנחנו מוסיפים את הקודקוד החדש לקוד המוקרן וכן את כל Control Nodes בדרך.

לאחר שהאנליזה הסתיימה אנחנו יוצרים שלושה סוגי פלטים:

1. רשימה מלאה של תוצאת האנליזה (analysis_result.txt)
 - a. טבלה המתארת איזה משתנים משפיעים ומושפעים עבור כל שורת קוד.
 - b. רשימת Control Edges
 - c. רשימת Dependency Edges
2. המחשה גרפית של הגרפים שהאנליזה יצרה:
 - a. Out.gv_control.pdf: גרף הControl Flow בתוכנית
 - b. Out.gv_dep.pdf: גרף הDependency בתוכנית
 - c. Out.gv_all.pdf: איחוד של שני הגרפים הקודמים
3. הקוד המוקרן על פני אחד המשתנים (projected_code.py)

איך להריץ?

התוכנית נבדקה ועובדת על פני פלטפורמת Windows, ופיית'ון 2.7.

החבילות הבאות צריכות להיות מותקנות על מנת שיהיה אפשר להריץ את התוכנית:

- Astor – חבילה שבאמצעותה אנחנו הופכים אובייקטי AST לשורת קוד:

<https://pypi.python.org/pypi/astor/0.5>

- Graphviz – חבילה שבאמצעותה אנחנו מדפיסים בצורה גרפית את הגרפים שאנחנו יוצרים:

<https://pypi.python.org/pypi/graphviz/0.4.8>

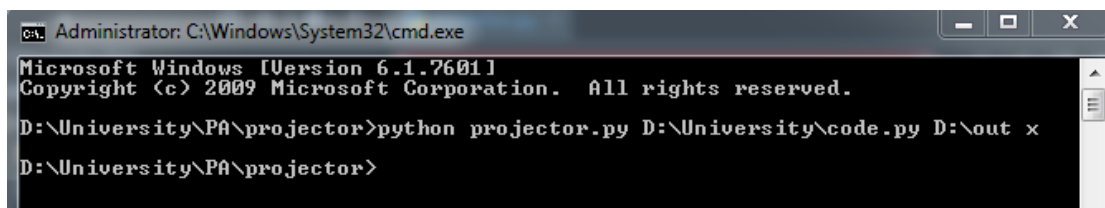
אם ברצונך להריץ את הטסטים, עליך להתקין pytest:

<https://pypi.python.org/pypi/pytest/2.8.5>

כאשר כל התלויות מוכנות, ניתן להריץ את האנליזה בצורה הבאה:

```
python projector.py <original_code> <output_dir> <projected variable>
```

דוגמאת הרצה:



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

D:\University\PA\projector>python projector.py D:\University\code.py D:\out x
D:\University\PA\projector>
```

על מנת להדגים את קריאת הפלט נריץ את האנליזה על התוכנית הבאה:

```
x = X()
if x > x:
    tmp = x
    if x > x:
        tmp.a = X()
    else:
        x.a = X()
else:
    tmp2 = x
    tmp2.a = X()
g = x
```

ההטלה שבחרתי להדגים היא על פני משתנה g.

ראשית נתרכז בטבלה שבפלט analysis_result.txt:

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1		['x', 'x']	if (x > x):
2	tmp	set(['x'])	tmp = x
3		['x', 'x']	if (x > x):
4	tmp#a	set(['tmp'])	tmp.a = X()
5			else:
6	x#a	set(['x'])	x.a = X()
7			else:
8	tmp2	set(['x'])	tmp2 = x
9	tmp2#a	set(['tmp2'])	tmp2.a = X()
10	g	set(['x', 'tmp#a', 'x#a', 'tmp2#a'])	g = x

ניתן לראות שישנה שורה עבור כל שורת קוד.

כל שורה מציגה איזה משתנה מושפע כתוצאה מהפעלת השורה (assigned variable), ואיזה משתנים משפיעים על הרצת השורה (influence variable). כך לדוגמא אם נסתכל על שורה 2, המשתנה המושפע הוא tmp כי זוהי שורת השמה למשתנה זה, והמשתנה המשפיע עליו הוא x. עבור שורה 1, אף משתנה לא מושפע מהרצת שורה זאת (זוהי שורת if), אך המשתנים המשפיע על שורה זאת הוא x.

כעת נתרכז ברשימות הקשתות המופיעות באותו קובץ:

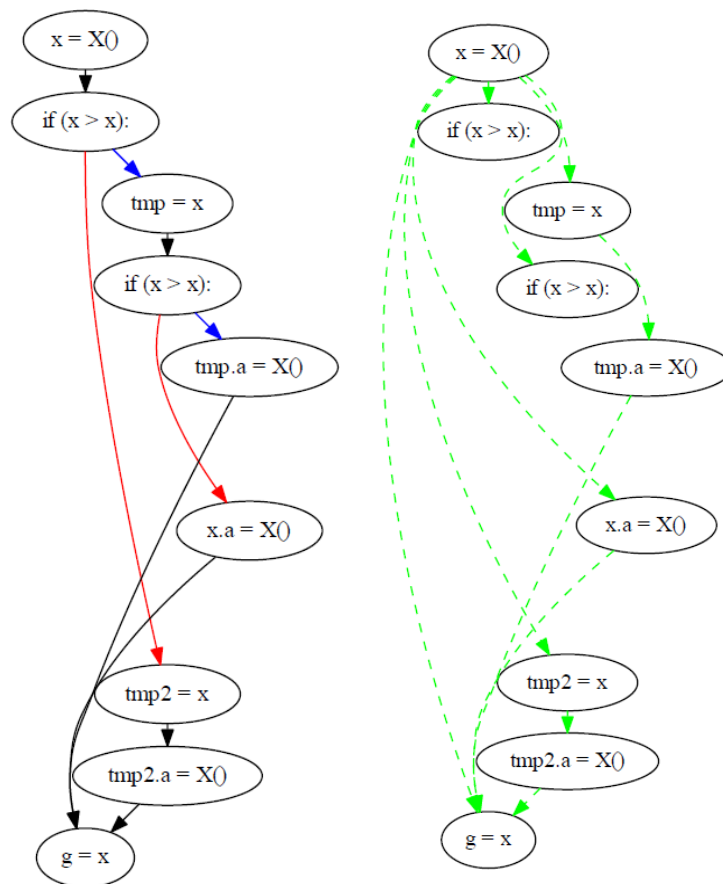
Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 10), (3, 6), (6, 10), (1, 8), (8, 9), (9, 10), (10, 11)]

Dependency Edges: [(0, 1), (2, 4), (0, 2), (0, 3), (0, 6), (8, 9), (0, 8), (0, 10), (4, 10), (6, 10), (9, 10)]

Control Edges מתארת את Control Flow של הקוד. אם הקוד טורי אזי יש קשת בין שורת קוד לשורה הבאה אחריה, לדוגמא 0 ל-1. אם יש קפיצות (לדוגמא ב if או while), אזי יש קשת עבור כל קפיצה מתאימה. לדוגמא 1 ל-2 ו 1 ל-8.

Dependency Edges היא רשימה המחזיקה את התלויות הלוגיות. לדוגמא זיהינו בשורה 10 את התלות בא, tmp.a, x.a, tmp2.a, ולכן יש קשת אל שורה 10 מההשמות אל משתנים אלו (נזכיר שאנחנו מניחים שהקוד בצורת SSA).

קבצי pdf שנוצרו בתיקיית הפלט מציגים את שני הגרפים בצורה גרפית:



ולבסוף ניתן לראות את הקוד המוקרן בקובץ projected_code.py.

הערה:

כפי שניתן לראות, הטבלה שבקובץ analysis_result הינה ליבת האנליזה, כל שאר הפעולות הן טכניות מעליה. לכן מעתה נציג במסמך רק את הטבלה הזאת.

כמו כן, הדוגמאות שנציג יהיו קצרות מאוד, כי אנחנו רוצים להתרכז במהות ולא לבלבל עם מספר רב של שורות קוד. ניתן להרחיב את הדוגמאות והקונספטים ליצירת קוד מורכב יותר.

- האנליזה היא intraprocedural.
- קוד הקלט הוא בצורת SSA (כיוון שאנחנו לא מאפשרים פונקציות, אזי אנחנו כן תומכים בהשמה למשתנה בשני branch שונים, ואנחנו מתחייבים להיות Sound אך לא מדויקים ביותר). אם הקוד אינו בצורת SSA יכול להיות שהאנליזה תעבוד כמו שצריך, אך אין הבטחה לכך.
- כל משתנה מחזיק אובייקט או Integer (ניתן להרחיב כך שהמשתנה יחזיק גם טיפוסים נוספים, אך זה עניין טכני).
- הקוד אינו מכיל הגדרות מכל סוג שהוא, ההנחה היא שהכל מוגדר כפי שצריך (לדוגמה אם קוראים לConstructor של המחלקה X אזי המחלקה מוגדרת כראוי).
- האנליזה מטפלת בהשמה של ערך או פעולה בינארית (ניתן להרחיב, זהו הליך טכני).
- הקוד יודע לטפל במבנה של if-else, while וקינון ביניהם.
- אובייקטים
 - האובייקטים הם מהצורה:

```
class MyObject(object):
    def __init__(self):
        pass
```

- כלומר בברירת מחדל ללא Attributes, ניתן להוסיף אותם דינמית.
- לא ניתן לגשת לAttribute של Attribute, לצורך כך יש להשתמש במשתנה עזר.
 - הפרויקט עובד על Shallow Pointers, לכן לדוגמה x.a.a משפיע על x.a ולא על x (בצורה דומה x.a משפיע על x).
 - הקריאה לConstructor חוקית.
 - דוגמה לסעיף אובייקטים:

נניח נתון הקוד הבא:

```
x = X()
x.a = X()
tmp = x.a
tmp.b = 2
x
x.a
```

אזי האנליזה תייצר את הטבלה הבאה עבורו:

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1	x#a	set(['x'])	x.a = X()
2	tmp	set(['x#a'])	tmp = x.a
3	tmp#b	set(['tmp'])	tmp.b = 2
4		set(['x', 'x#a'])	x
5		set(['tmp#b', 'x#a'])	x.a

בחלק זה נתאר את האלגוריתם כאשר אין Aliasing. את שדרוג האלגוריתם על מנת שיתמוך בAliasing נפרט בחלק הבא.

אנחנו מתארים את הקוד רק מבחינה לוגית על מנת לשמור על הסדר וההבנה. מבחינה טכנית הקוד מסובך בהרבה ויש בו עוד פעולות רבות.

עבור כל שורת קוד שאנחנו מגיעים אליה:

- עבור שורת השמה:
 - מצא את המשתנה שההשמה מתבצעת אליו
 - מצא אילו משתנים משפיעים על ההשמה:
 - בפעולה בינארית – כל אחד מהמשתנים המשתתפים בפעולה הבינארית
 - בהשמת משתנה – המשתנה עצמו וכל השמה לAttribute'ים שלו
 - בהשמת Attribute – Attribute עצמו וכל השמה לAttribute'ים שלו
 - אם משימים לAttribute – המשתנה עצמו
 - צור קודקוד חדש המייצג את הצומת
 - נסה ליצור קשתות dependency בין השורות בהם מבצעים השמה למשתנים המשפיעים לבין השורה הנוכחית:
 - אנחנו עושים זאת באמצעות מילון המחזיק עבור כל משתנה מתי הוא הושם לאחרונה. המילון יכול למפות למספר שורות קוד שונות, כפי שהוסבר בהנחות העבודה.
 - ייתכן שהמשתנה המשפיע אינו נמצא במילון. זה קורה בגלל שאנחנו בקטע קוד פנימי, והמשתנה הוגדר בבלוק החיצוני. אנחנו מכניסים את המשתנה הנעלם למילון נוסף, ומסמנים על איזה שורות הוא משפיע.
 - צור קשתות Control לצומת הבאה.
 - עדכן את המילון המחזיק לכל משתנה מתי הוא הושם בפעם האחרונה.
- עבור שורת ביטוי:
 - הטיפול מתבצע בצורה דומה פרט לכך שלא מוצאים את המשתנה שההשמה מתבצעת אליו ולא מעדכנים את המילון
- עבור שורת IF:
 - צור קשתות תלות בין התנאי שבIF וההשמות של המשתנים הרלוונטיים
 - צור קודקוד Control
 - צור קשת עבור Then ועבור elsen
 - עבור החלק של Then וה Else:
 - הרץ את האנליזה בצורה רקורסיבית על החלקים הפנימיים
 - אחד בין תוצאת האנליזה הפנימית לריצה הנוכחית:
 - אחד את רשימת הקודקודים
 - אחד את רשימת הקשתות
 - נסה לטפל במשתנים לא ידועים של הקוד הפנימי
 - תקן קשתות שמצביעות לא למקום הנכון (לדוגמא אם יש בלוק if-else בתוך if אז הסוף של if הפנימי עלול לעשות קשת ממנו לקוד שאחרי elsen הפנימי, למרות שהוא צריך להצביע לקוד שאחרי elsen החיצוני).
 - עשה merge למילון ששומר מתי משתנה נראה לאחרונה
- עבור שורת While:
 - הטיפול דומה לטיפול בIF
 - השוני הוא שכיוון והריצה הראשונה של הלולאה יכולה להשפיע על הריצה השניה אזי מריצים את האנליזה פעמיים על קטע הקוד ומעדכנים את המילונים בהתאם.

דוגמאת קוד:

```

x = 2
t = 124
counter = 0
while t < x:
    t = t + 5
    x = 2
    t = t + 5
    counter = counter + 1
    if t > x:
        t = t - counter
        counter = counter + x
    else:
        x = x + 100
        counter = counter - 1
        t = counter + x

```

תוצאת האנליזה:

#line	assigned variable	influence variables	statement
0	x	set([])	x = 2
1	t	set([])	t = 124
2	counter	set([])	counter = 0
3		['t', 'x']	while (t < x):
4	t	set(['t'])	t = (t + 5)
5	x	set([])	x = 2
6	t	set(['t'])	t = (t + 5)
7	counter	set(['counter'])	counter = (counter + 1)
8		['t', 'x']	if (t > x):
9	t	set(['counter', 't'])	t = (t - counter)
10	counter	set(['x', 'counter'])	counter = (counter + x)
11			else:
12	x	set(['x'])	x = (x + 100)
13	counter	set(['counter'])	counter = (counter - 1)
14	t	set(['x', 'counter'])	t = (counter + x)
15		set(['t'])	t

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10), (10, 3), (8, 12), (12, 13), (13, 14), (14, 3), (3, 15), (15, 16)]

Dependency Edges: [(1, 3), (0, 3), (4, 6), (6, 8), (5, 8), (5, 10), (7, 9), (7, 10), (6, 9), (12, 14), (13, 14), (5, 12), (7, 13), (2, 7), (1, 4), (10, 7), (13, 7), (14, 4), (9, 4), (14, 3), (9, 3), (12, 3), (5, 3), (1, 15), (14, 15), (9, 15)]

הסבר:

- על שורות 0-3 אין השפעות חיצוניות. כאשר הקוד מטפל בכל אחת מהן הוא מעדכן מתי הוא ראה את כל אחד מהמשתנים הללו לאחרונה.
- התנאי של הwhile הוא $t < x$, לכן t ו- x משפיעים ומתווספות קשתות (1, 3) ו-(0, 3).
- עבור הריצה הראשונה של הלולאה מפעילים את האנליזה בצורה רקורסיבית, כעת הקוד הנבדק הוא גוף הלולאה.
 - השורה $t=t+5$ מושפעת מה t החיצוני. אכן נוצרת הקשת (1,4), אך כפי שניתן לראות היא נוצרת בשלב מאוחר יחסית, כי כשאנחנו עובדים על הקוד הפנימי אנחנו לא יודעים מיהו t ואנחנו מכניסים אותו בהתחלה ל"רשימת המשתנים הלא ידועים". מעדכנים מתי שמנו ערך פעם אחרונה ל- t .
 - השורה $x=2$ לא מושפעת מכלום, מעדכנים מתי שמנו ערך ל- x .
 - השורה $t=t+5$ (6) כעת מושפעת מה t בשורה 4, לכן נוצרת הקשת (4, 6) ואנחנו מעדכנים את המילון.
 - $\text{Counter} = \text{counter} + 1$ טיפול דומה לשורה הראשונה $t=t+5$
 - מטפלים ב-`if` וב-`else` בנפרד אך בצורה דומה לדרך שבה אנחנו מטפלים ב-`While`. לאחר מכן ממזגים את תוצאת האנליזה שלהם לגוף הwhile:
 - לאחר פעולת המיזוג t הושם לאחרונה בשני מקומות שונים – 14 ו-9. גם Counter הושם לאחרונה בשני מקומות – 10 ו-13.
 - מעלים את תוצאת האנליזה של הwhile block וממזגים אותה עם הבלוק הראשי. נשים לב שעכשיו לדוגמא t נראה ב-3 מקומות שונים – 1, 9 ו-14. (אם היינו יודעים בוודאות שלולאת הwhile מתבצעת אזי היינו יכולים לשכוח מההשמה בשורה 1, אך אנחנו לא יודעים זאת).
 - כמו כן, אנחנו יוצרים קשתות עבור כל המשתנים שלא ידענו היכן הוגדרו. יכול להיות שגם בבלוק החיצוני המשתנים הללו לא ידועים, ובמקרה כזה נסמן את המשתנים ככאלו, ונחכה לבלוק חיצוני יותר שיטפל בהם).
- עבור כל ריצת לולאה נוספת אנחנו מבצעים את האנליזה על גוף הלולאה פעם נוספת. כך לדוגמא נוצרות הקשתות (9, 4) ו-(14,4).

על מנת שנוכל לזהות Aliasing אנחנו משתמשים בPoints-to Analysis עם מודיפיקציה קטנה לאבסטרקציה שנלמדה בכיתה.

תזכורת: בשיעור למדנו שהDomain האבסטרקטי הוא רשימתה זוגות כאשר האיבר הראשון בזוג הוא משתנה המצביע למשתנה שהוא האיבר השני בזוג.

באנליזה שלנו, הDomain הוא רשימה של זוגות כאשר האיבר הראשון הוא משתנה והאיבר השני מייצג את האובייקט שאליו האיבר הראשון מצביע.

בחרנו לממש זאת באמצעות שני מילונים:

- `Var_to_object`: מילון שהמפתח שלו הוא שם משתנה והvalue הוא רשימה של האובייקטים הפוטנציאליים אליהם המשתנה מצביע.
- `Object_to_var`: מילון שהמפתח שלו הוא שם אובייקט והvalue הוא רשימה של משתנים פוטנציאליים שמצביעים אליו.

כאשר אנחנו מבצעים פעולת השמה:

- אם יוצרים אובייקט חדש (קריאה לConstructor) – אנחנו יוצרים ערך חדש בשני המילונים המצמד בין האובייקט שנוצר למשתנה.
- אם משימים משתנה:
 - אם המשתנה שאנחנו משימים הוגדר כחלק מהבלוק – מצא על אילו אובייקטים המשתנה מצביע:
 - עדכן את `var_to_object` עם ערך חדש שבו הkey הוא המשתנה שאליו משימים והvalue הוא רשימת האובייקטים שמצאנו
 - הוסף ל`Object_to_var` עבור כל אובייקט פוטנציאלי את המשתנה אליו משימים.
 - אם המשתנה הוגדר כחלק מבלוק חיצוני – בבלוק הפנימי אין לנו את המידע לאן הוא מצביע, עדכן את המילונים עם משתנה דמה שנחפש ונעדכן כחלק מתהליך האיחוד של הבלוק הפנימי לחיצוני.

כאשר מבצעים פעולת קריאה:

- נשתמש במילונים על מנת למצוא את כל הattributes של האובייקט שאליו אנחנו מצביעים, גם כאשר השינוי נעשה דרך

בפרק זה נציג מספר דוגמאות שמציגות את הכח של האנליזה. כפי שהוסבר קודם, הדוגמאות פשוטות על מנת לשמור אותן מובנות, ולהדגים את העקרונות הראשיים. ניתן להגדיל ולסבך את הקוד כרצוננו. כמו כן, בכל דוגמא נציג את אחד Output שמדגים בצורה הכי ברורה את העקרונות. המעבר מOutput אחד לאחר הוא עניין טכני כפי שהוסבר בפרקים הקודמים.

1. בדוגמא הבאה אנחנו משנים את האובייקט alias שלו, ומצפים שהאנליזה תזהה את השינוי באובייקט:

```
x = X()
tmp = x
tmp.a = X()
x
```

תוצאת האנליזה:

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1	tmp	set(['x'])	tmp = x
2	tmp#a	set(['tmp'])	tmp.a = X()
3		set(['x', 'tmp#a'])	x

כפי שניתן לראות בשורה 3, האנליזה מזהה שההשמה לtmp.a משנה את האובייקט אליו x מצביע.

2. בדוגמא הבאה אנחנו יוצרים alias למשתנה ומצפים שהאנליזה תכיר את attribute שיצרנו דרך המשתנה המקורי:

```
x = X()
x.a = X()
tmp = x
tmp.a
```

תוצאת האנליזה:

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1	x#a	set(['x'])	x.a = X()
2	tmp	set(['x', 'x#a'])	tmp = x
3		set(['tmp', 'x#a'])	tmp.a

האנליזה מזהה שtmp.a תלוי בtmp.x.a.

3. עדכון attribute פנימיים:

```

x = X()
y = Y()
x.a = y
tmp = x.a
tmp.c = C()
y.b = Z()
x
x.a

```

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1	y	set([])	y = Y()
2	x#a	set(['y', 'x'])	x.a = y
3	tmp	set(['x#a'])	tmp = x.a
4	tmp#c	set(['tmp'])	tmp.c = C()
5	y#b	set(['y'])	y.b = Z()
6		set(['x', 'x#a'])	x
7		set(['tmp#c', 'y#b', 'x#a'])	x.a

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8)]

Dependency Edges: [(1, 2), (0, 2), (2, 3), (3, 4), (1, 5), (0, 6), (2, 6), (4, 7), (5, 7), (2, 7)]

נסתכל ראשית על המשתנה x בשורה 6. הוא תלוי בשורה 0 שם הושם לו ערך לראשונה. לכן קיימת הקשת (0, 6). כמו כן, למשתנה x יש attribute בודד – x.a. attribute זה נוצר בשורה 2, לכן קיימת הקשת (2, 6). כפי שהוסבר, אנחנו לא מעוניינים שattribute של attribute ישפיעו לנו על האובייקט הראשי.

נשים לב שאחרי שורה 3 x.a, tmp ו-y מצביעים לאותו אובייקט. לאובייקט זה אנחנו יוצרים שני attribute, פעם אחת באמצעות tmp.c ופעם נוספת באמצעות y.b. לכן למשתנה x.a יש 3 תלויות – y.b (לכן נוצרת הקשת (5,7)), tmp.c (לכן נוצרת הקשת (4,7)), וליצירה בפעם הראשונה של attribute (לכן נוצרת הקשת (2, 7)).

4. Attribute שונים של אותו משתנה לא משפיעים אחד על השני:

```

y = Y()
y.b = Y()
y.c = Y()
y.b

```

#line	assigned variable	influence variables	statement
0	y	set([])	y = Y()
1	y#b	set(['y'])	y.b = Y()
2	y#c	set(['y'])	y.c = Y()
3		set(['y#b'])	y.b

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4)]

Dependency Edges: [(0, 1), (0, 2), (1, 3)]

כפי שניתן לראות, הattributen y.c לא משפיע על y.b.

5. שינוי attribute משני מקומות (if-else):

```

a = 1
b = 2
x = X()
y = Y()
z = Z()
if a > b:
    x.a = y
else:
    x.a = z
x.a

```

#line	assigned variable	influence variables	statement
0	a	set([])	a = 1
1	b	set([])	b = 2
2	x	set([])	x = X()
3	y	set([])	y = Y()
4	z	set([])	z = Z()
5		['a', 'b']	if (a > b):
6	x#a	set(['y', 'x'])	x.a = y
7			else:
8	x#a	set(['x', 'z'])	x.a = z
9		set(['x#a'])	x.a

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 9), (5, 8), (8, 9), (9, 10)]

Dependency Edges: [(0, 5), (1, 5), (3, 6), (2, 6), (2, 8), (4, 8), (8, 9), (6, 9)]

כפי שניתן לראות, x.a בשורה 9 מושפע הן מההשמה בשורה 8 והן מההשמה בשורה 6, לכן נוספות שתי קשתות – (6,9), (8, 9).

.6

```

x = X()
if x > x:
    tmp = x
    if x > x:
        tmp.a = X()
    else:
        x.a = X()
else:
    tmp2 = x
    tmp2.a = X()
x

```


#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1		['x', 'x']	if (x > x):
2	tmp	set(['x'])	tmp = x
3		['x', 'x']	if (x > x):
4	tmp#a	set(['tmp'])	tmp.a = X()
5			else:
6	x#a	set(['x'])	x.a = X()
7			else:
8	tmp2	set(['x'])	tmp2 = x
9	tmp2#a	set(['tmp2'])	tmp2.a = X()
10		set(['x', 'tmp#a', 'x#a', 'tmp2#a'])	x

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 10), (3, 6), (6, 10), (1, 8), (8, 9), (9, 10), (10, 11)]

Dependency Edges: [(0, 1), (2, 4), (0, 2), (0, 3), (0, 6), (8, 9), (0, 8), (0, 10), (4, 10), (6, 10), (9, 10)]

הדוגמא מראה מספר צורות לביצוע Aliasing.

1. Aliasing המתבצע בתוך בלוק פנימי (שורות 8-9), שורה 10 מזהה את הנגיעה מ tmp2.a ולכן נוצרת הקשת (9, 10).
2. Aliasing המתבצע מבחן חיצוני (שורות 2-6), האlias מתבצע מחוץ לבלוק של ה if-else הפנימי. הקוד מזהה את הנגיעה ב tmp.a. (נוצרה קשת (4, 10))
3. ללא Aliasing, שורה 6, הקוד מזהה את הנגיעה ויוצר קשת (6, 10).

7. דוגמא לאיבוד דיוק:

```

x = X()
y = Y()
if x > y:
    tmp = x
else:
    tmp = y
tmp.a = 2
x

```

#line	assigned variable	influence variables	statement
0	x	set([])	x = X()
1	y	set([])	y = Y()
2		['x', 'y']	if (x > y):
3	tmp	set(['x'])	tmp = x
4			else:
5	tmp	set(['y'])	tmp = y
6	tmp#a	set(['tmp'])	tmp.a = 2
7		set(['x', 'tmp#a'])	x

Control Edges: [(0, 1), (1, 2), (2, 3), (3, 6), (2, 5), (5, 6), (6, 7), (7, 8)]

Dependency Edges: [(0, 2), (1, 2), (0, 3), (1, 5), (3, 6), (5, 6), (0, 7), (6, 7)]

בדוגמא זאת ברור כי x לא יכול להיות מושפע מהשפעת else אך כן נוצרת קשת (6, 7), זה קורה כי קוד המקור אינו SSA.

יחד עם זאת חשוב להדגיש שאנחנו Sound, כלומר לעולם לא נפספס השמות, אך אולי נייצר קשתות מיותרות.

דוגמאות נוספות בדוקות ועובדות ניתן למצוא בקובץ TestAll.py.