Pointer Analysis

Zahra Hosseini, Iowa State University 15/02/2016

1 Introduction

A pointer analysis attempts to statically determine the possible run-time value of a pointer.A pointer alias analysis attempts to determine when two pointer expressions refer to the same storage location. [1]
In this project, we try to implement Anderson's Algorithm for Points-to Analysis
The Basic algorithm for C is like following:

1. $x = \&y \mid y \in pt(x)$
2. $x = y \mid pt(y) \subseteq pt(x)$
3. $x = *y \mid \forall a \in pt(y).pt(a) \subseteq pt(x)$
4. $*x = y \mid \forall w \in pt(x).pt(y) \subseteq pt(w)$

Because in Java we don't have item 1,3,4 we need to implement item 2 which is Refrence to Refrence in Java.

2 Algorithm

This Implementation is **flow-insensitive** and **context-insensitive**.
Based on [2], **Field-Sensitivity** added to this project.
Field Sensitivity: Based on Anderson Algorithm, add object reference fields to points-to graph as suffices for reference variables. For example: class A has fields f,g then $p = newA()$, means p.f and p.g are in the points-to graph[3]

Statement	Constraint	
`i: x = new T()`	$\{o_i\} \subseteq pt(x)$	[New]
`x = y`	$pt(y) \subseteq pt(x)$	[Assign]
`x = y.f`	$\frac{o_i \in pt(y)}{pt(o_i.f) \subseteq pt(x)}$	[Load]
`x.f = y`	$\frac{o_i \in pt(x)}{pt(y) \subseteq pt(o_i.f)}$	[Store]

Table 1. Canonical statements for Java points-to analysis and the corresponding points-to set constraints.

This implementation is **Subset-based**: The analysis models directionality of assignments; i.e., a statement $x = y$ implies $pt(y) \subseteq pt(x)$.
Implemented field-sensitivity is like bellow:

1. $x = y | \ pt(y) \subseteq pt(x)$
2. $y.f = x \mid \forall o \in pt(y)(pt(x) \subseteq pt(o.f))$
3. $x = y.f \mid \forall o \in pt(y)(o.f \subseteq pt(x))$

The abstract idea of implementation part is like following:

Listing 1: Implemented Anderson Algorithm

```
initialize graph and PointsTo set
W={v|ptr(v)!=empty}
while (!W.isempty)
    v=W.remove(0);
    foreach edge new->q
        ptr(q)=ptr(q)+newID;
        add q to w
    foreach edge v->q
        ptr(q)=ptr(q)+ptr(v);
        add q to w if q changed
    foreach edge v.f->q
        ptr(q)=ptr(q)+ptr(v.f);
        add q to w if q changed
    foreach edge v->q.f
        ptr(q.f)=ptr(q.f)+ptr(v);
        add q.f to w if q.f changed
```

3 How to RUN

For Running this project, we need to put TestCase in /src/TestCase directory and in StandaloneMain.java

Listing 2: StandaloneMain.java .

```
1     //Change to Name of Class
2     //TestCase is name of package
3         String ClassName = "TestCase.Elevator";
4     //Change to name of file which we want to analyse
5         String LocationoftestFile = "src\\TestCase\\Elevator.java";
```

After running successfully, two output file will created in main directory of project which is the final result based on problem definition.

4 Results

The result is consist of two file which named output1.txt and output2.txt
Output1.txt: Consists of Line of Code, performance, Average set of points-to set for each method
output2.txt: File name, Line number, Variable, Points-to set
Because my program is context-insensitive, it needs to set that we want to analyse which method of my program, I create a loop, so it will iterate over all methods in the program which is given.Thus,I add name of method-call for Output1 and Output2.txt is also consist several line related to each method.

5 Real-world program

I use Software-artifact Infrastructure Repository (SIR) for experimentation part of this project. I download two Java projects (AlarmClock and Elevator) and run my project to find points-to based on Anderson's Algorithm with adding field sensitivity to it.
The repository contains many Java and C, $C++$, and $C\#$ software systems, in multiple versions, together with supporting artifacts such as test suites, fault data, and scripts.[4]

References

[1] Michael Hind. Pointer analysis: Haven't we solved this problem yet? In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '01, pages 54–61, New York, NY, USA, 2001. ACM.

[2] Manu Sridharan and Stephen J. Fink. *Static Analysis: 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings*, chapter The Complexity of Andersen's Analysis in Practice, pages 205–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] Atanas Rountev, Ana Milanova, and Barbara G. Ryder. Points-to analysis for java using annotated constraints. *SIGPLAN Not.*, 36(11):43–55, October 2001.

[4] Software artifact Infrastructure Repository. Software-artifact infrastructure repository, 2016.