# Null Dereference Analysis

E0 227 : Program Analysis and Verification, 2018

Stanly Samuel and Rekha Pai
*Instructors:* Prof. K. V. Raghavan, Prof. D. D'Souza
*Computer Science and Automation,*
*Indian Institute of Science, Bangalore*

## Goal

*Given a Java program, implement a tool that performs:*
*1) Intraprocedural pointer analysis using Algorithm A.*
*2) Interprocedural pointer analysis using Algorithm A + Iterative Method.*

## The Programming Language

- Your analysis should work on any *arbitrary* Java program. All syntactic features of Java ought to be handled with assumptions given below.

- In **Phase I**, for simplicity,

– **Method calls returning reference to objects** will be considered as allocation sites. (name the object as newi where 'i' is the line number of the method call)

– All other method calls to be treated as no-ops.

– Treat exception edges as identity transfer functions.

–Wala converts constructor calls to calls to a special static method. Treat these also as identity transfer functions.

# Pointer Analysis (An Introduction)

Gives information about what objects are pointed to by which pointer variables

```
void foo {
1: temp1=null;
2: temp2=new;
3: temp3= temp2;
}
```

temp1 points to null

temp2 points to obj created at site s2

temp3 points to obj pointed to by temp2

1: temp1-> {NULL}

2: temp1-> {NULL},
temp2->{new2}

3: temp1-> {NULL},
temp2->{new2},
temp3->{new2}

new2 represents all symbolic objects created at line 2

# Example 2

```
1: void foo(){
2:     temp1=new ;
3:     if(i%2==0){  //Assume i to be a random integer.
4:            temp1=null;
5:     }
6:     temp2=temp1;
       // temp2 may point to O2 or NULL here.
7: }
```

2: temp1->{ new2}

4:  temp1->{NULL}

5: temp1->{new2,NULL}

6: temp1->{new2,NULL}, temp2->{ new2,NULL}

# Example 3 (interprocedural case)

```
 /* consider temp1, temp2 are global
 and do not point to anything yet
 */
 2: void foo(){
3:              temp1 = null;
4:              bar();
5:              temp2=temp1;
6:     }
7:     void bar() {
8:                       temp1=new();
9:                       return;
10:    }
```

Analysis of method foo:

3: temp1->{NULL}

5: temp1->{new8}, temp2->{new8}

8: temp1->{new8}

## Implementation Details

## WALA

- IBM T. J. WAtson Libraries for Analysis
- Framework for analysis of Java programs
- WALA, like other tools, converts Java byte code to an “intermediate representation”, which is more amenable to analysis
- Your analysis take an Intermediate Representation (IR) as the input.

# Sample 1 – WALA Intermediate Representation

Static Single Assignment (SSA)

```
1: public class SampleTests {
2:
3:     public static void foo(int i) {
4:             SampleTests t1 = new
SampleTests();
5:             SampleTests t2 = new
SampleTests();
6:             SampleTests t3 = null;
7:             if(i>10) {
8:                     t3 = t1;
9:             } else {
10:                    t3 = t2;
11:     }
12:     t3.toString();
13: }
```

```
BB0
BB1
0   v3 = new <Application,LTestCases/SampleTests>@0(line 4)
BB2
2   invokespecial < Application, LTestCases/SampleTests, <init>()V > v3 @4
exception:v4(line 4)
BB3
4   v5 = new <Application,LTestCases/SampleTests>@8(line 5)
BB4
6   invokespecial < Application, LTestCases/SampleTests, <init>()V > v5 @12
exception:v6(line 5)
BB5
12   conditional branch(le, to iindex=16) v1,v8:#10(line 7) {1=[i]}
BB6
15   goto (from iindex= 15 to iindex = 18)   (line 9)
BB7
BB8
         v9 = phi  v3,v5
19   v11 = invokevirtual < Application, Ljava/lang/Object,
toString()Ljava/lang/String; > v9 @32 exception:v10(line 12) {9=[t3]}
BB9
21   return                         (line 13)
BB10
```

# Sample 1 – WALA Intermediate Representation

Control Flow Graph Nodes

```
1: public class SampleTests {
2:
3:     public static void foo(int i) {
4:             SampleTests t1 = new
SampleTests();
5:             SampleTests t2 = new
SampleTests();
6:             SampleTests t3 = null;
7:             if(i>10) {
8:                     t3 = t1;
9:             } else {
10:                    t3 = t2;
11:    }
12:    t3.toString();
13: }
```

**BB0**
**BB1**
0 v3 = new <Application,LTestCases/SampleTests>@0(line 4)
**BB2**
2 invokespecial < Application, LTestCases/SampleTests, <init>()V > v3 @4 exception:v4(line 4)
**BB3**
4 v5 = new <Application,LTestCases/SampleTests>@8(line 5)
**BB4**
6 invokespecial < Application, LTestCases/SampleTests, <init>()V > v5 @12 exception:v6(line 5)
**BB5**
12 conditional branch(le, to iindex=16) v1,v8:#10(line 7) {1=[i]}
**BB6**
15 goto (from iindex= 15 to iindex = 18) (line 9)
**BB7**
**BB8**
v9 = phi v3,v5
19 v11 = invokevirtual < Application, Ljava/lang/Object, toString()Ljava/lang/String; > v9 @32 exception:v10(line12) {9=[t3]}
**BB9**
21 return (line 13)
**BB10**

# Sample 1 – WALA Intermediate Representation

Phi Nodes

```
1: public class SampleTests {
2:
3:     public static void foo(int i) {
4:             SampleTests t1 = new
SampleTests();
5:             SampleTests t2 = new
SampleTests();
6:             SampleTests t3 = null;
7:             if(i>10) {
8:                     t3 = t1;
9:             } else {
10:                    t3 = t2;
11:    }
12:    t3.toString();
13: }
```

```
BB0
BB1
0   v3 = new <Application,LTestCases/SampleTests>@0(line 4)
BB2
2   invokespecial < Application, LTestCases/SampleTests, <init>()V > v3 @4
exception:v4(line 4)
BB3
4   v5 = new <Application,LTestCases/SampleTests>@8(line 5)
BB4
6   invokespecial < Application, LTestCases/SampleTests, <init>()V > v5 @12
exception:v6(line 5)
BB5
12   conditional branch(le, to iindex=16) v1,v8:#10(line 7) {1=[i]}
BB6
15   goto (from iindex= 15 to iindex = 18)   (line 9)
BB7
BB8
        v9 = phi  v3,v5
19   v11 = invokevirtual < Application, Ljava/lang/Object,
toString()Ljava/lang/String; > v9 @32 exception:v10(line 12) {9=[t3]}
BB9
21   return                              (line 13)
BB10
```

# Sample 1 – WALA CFG

```
public class SampleTests {

    public static void foo(int i) {
        SampleTests t1 = new SampleTests();
        SampleTests t2 = new SampleTests();
        SampleTests t3 = null;
        if(i>10) {
            t3 = t1;
        } else {
        t3 = t2;
    }
    t3.toString();
}
```

```
BB0[-1..-2]
   -> BB1
BB1[0..0]
   -> BB2
   -> BB10
BB2[1..2]
   -> BB3
   -> BB10
BB3[3..4]
   -> BB4
   -> BB10
BB4[5..6]
   -> BB5
   -> BB10
BB5[7..12]
   -> BB7
   -> BB6
BB6[13..15]
   -> BB8
BB7[16..17]
   -> BB8
BB8[18..19]
   -> BB9
   -> BB10
BB9[20..21]
   -> BB10
BB10[-1..-2]
```

# Understanding WALA IR: Statements

```
0   v3 = new <Application,LTestCases/SampleTests>@0(line 4)
```

Line number in IR

Name of new variable

Type of the variable

Line number in source code

```
2   invokespecial < Application, LTestCases/SampleTests, <init>()V > v3 @4 exception:v4(line 4)
```

Call to constructor of v3

```
12  conditional branch(le, to iindex=16) v1,v8:#10(line 7) {1=[i]}
```

less than or equal to

v1 <= v8

v8 is a constant 10

v1 corresponds to i in source code

```
15   goto (from iindex= 15 to iindex = 18)   (line 9)
```

Jumps to BB containing this line number.
Note that this line number might not be displayed in the printed IR

## Understanding WALA IR: Basic Blocks

$$\begin{tikzcd} & \text{BB2[1..2]} \arrow[dl] \arrow[dr] & \\ \text{Instruction number of first instruction in this BB} & & \text{Instruction number of last instruction in this BB} \end{tikzcd}$$

Note: These instructions might not be visible in the IR

If a BB contains multiple outgoing edges and one of it is an outgoing edge to the last BB then there is a high possibility that that edge is due to a possible exception that can be thrown. You can check which edges are exception edges from the CFG using WALA APIs (SSACFG.hasExceptionalEdge and SSACFG.getExceptionalSuccessors)

```
BB1[0..0]
   -> BB2
   -> BB10
```

```
BB1
0   v3 = new <Application,LTestCases/SampleTests>@0(line 4)
```

For this example, BB1 has an outgoing edge to BB2 and BB10. The edge BB1 -> BB10 corresponds to the exception that is thrown when a new object is allocated (if allocation fails due to any reason).

# Understanding WALA IR: Phi nodes and conditionals

```
BB5[7..12]
   -> BB7
   -> BB6
```

```
BB5
12   conditional branch(le, to iindex=16) v1,v8:#10(line 7) {1=[i]}
```

Outgoing edge of BB5 corresponding to the true branch of the condition goes to BB7 and the false branch to BB6

```
BB6[13..15]
   -> BB8
BB7[16..17]
   -> BB8
BB8[18..19]
   -> BB9
   -> BB10
```

```
BB6
15   goto (from iindex= 15 to iindex = 18)   (line 9)
BB7
BB8
          v9 = phi  v3,v5
19   v11 = invokevirtual < Application, Ljava/lang/Object, toString()Ljava/lang/String; > v9 @32 exception:v10(line 12) {9=[t3]}
```

From the CFG it is clear that BB8 is the merge node of the conditional in BB5. Thus it contains the phi instruction. In the phi instruction, v3 corresponds to the value that comes from the false branch of BB5 (i.e. BB6) and v5 to the true branch (i.e. BB7). This is in reverse with respect to the order shown in the CFG for BB5. Thus if BB5 has a deterministic conditional, then you need to be careful as to which variable v9 should be assigned to. Note that this explanation is valid only for this example. Getting the predecessor nodes of a BB will return it in the order in which the phi node's variables are arranged. That is the first predecessor corresponds to the first variable and so on.

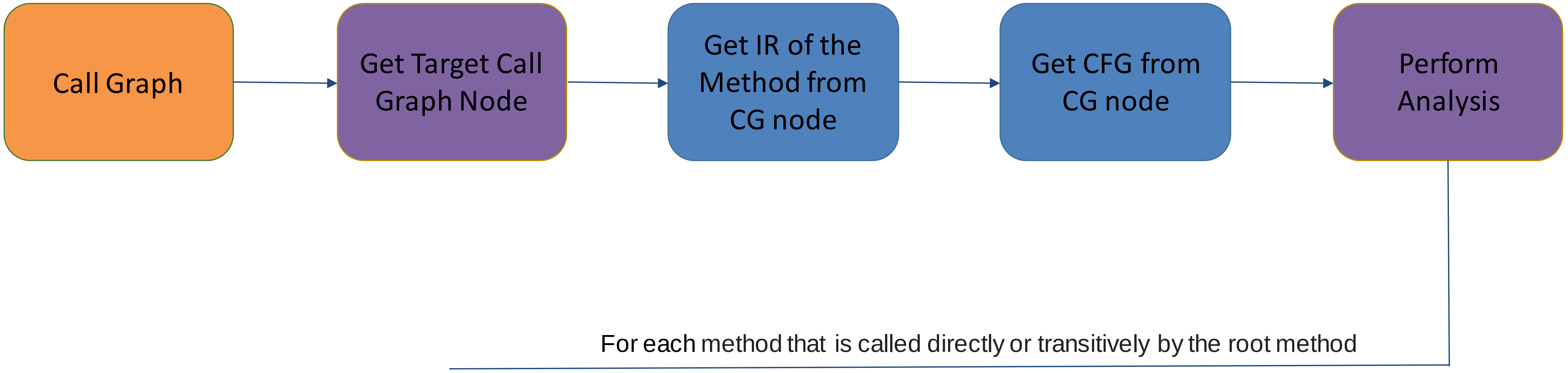
## Analysis Workflow – Setting Up

- Call graph – A directed graph in which nodes are the subroutines and an edge from node $n_1$ to node $n_2$ represents that $n_1$ calls $n_2$
- Initial Set-up –
  - Code provided to set up the analysis
  - Class – SetUpAnalysis.java
  - Call graph of the entire JAR and IR of the method to be analysed are provided
  - printnodes() – prints the nodes in the call graph
  - printIR() - prints the IR of the given method
- No need to change the provided code

# Analysis Workflow – Providing Arguments

- Main Class: PAVNullDerefAnalysis.java
- Input (Program Arguments)
  - args[0]: path to the application jar
  - args[1]: fully qualified name of the main class. Fully qualified name has the format L<package_name>/<class_name>
  - args[2]: fully qualified name of the class containing the method to be analyzed
  - args[3]: root method to be analyzed
  - E.g. the arguments to analyse foo in Sample 1 would be:
- <path_to_jar> LTestCases/SampleTests LTestCases/SampleTests foo(I)V

# Analysis Workflow – What you need to do


Call Graph
Get Target Call Graph Node
Get IR of the Method from CG node
Get CFG from CG node
Perform Analysis
For each method that is called directly or transitively by the root method

## Analysis Workflow – What You Need To Do

- Devise a suitable data structure for the points-to graph
- For computing the points-to sets,
  - ➢ you need to maintain a table of points-to graph at every program point in the CFG
  - ➢ you should design the lattice as well as design the transfer functions yourself
- For Interprocedural Analysis,
  - ➢ use the iterative method.
  - ➢ design the transfer functions for the call and return nodes.
- The analysis needs to be flow sensitive.
- Write the output to a file in the specified format
- The requirements for Phase II will be specified later

# Analysis Workflow – The Classes

```
public class PAVNullDerefAnalysis {

        private SetUpAnalysis setup;

        public PAVNullDerefAnalysis (String classpath, String mainClass, String analysisClass, String analysisMethod) {
                setup = new SetUpAnalysis(classpath, mainClass, analysisClass, analysisMethod);
        }

        public static void main(String[] args) throws Exception {
                String classpath, mainClass, analysisClass, analysisMethod;

                classpath = args[0];
                mainClass = args[1];
                analysisClass = args[2];
                analysisMethod = args[3];

                PAVNullDerefAnalysis  pAnalysis = new PAVNullDerefAnalysis (classpath, mainClass, analysisClass, 
analysisMethod);
                pAnalysis.runAnalysis();

        }
}
```

# Sample 1 – Input Code with IR

arguments: <path_to_jar> LTestCases/SampleTests LTestCases/SampleTests foo(I)V

```
1: public class SampleTests {
2:
3:     public static void foo(int i) {
4:             SampleTests t1 = null;
5:             SampleTests t2 = new
SampleTests();
6:             SampleTests t3 = null;
7:             if(i>10) {
8:                     t3 = t1;
9:             } else {
10:            t3 = t2;
11:    }
12:    t3.toString();
13: }
```

```
BB0
BB1
2   v4 = new <Application,LTestCases/SampleTests>@2(line 13)
BB2
4   invokespecial < Application, LTestCases/SampleTests, <init>()V > v4 @6
exception:v5(line 13)
BB3
10   conditional branch(le, to iindex=14) v1,v6:#10(line 15) {1=[i]}
BB4
13   goto (from iindex= 13 to iindex = 16)   (line 17)
BB5
BB6
        v7 = phi  v3:#null,v4
17   v9 = invokevirtual < Application, Ljava/lang/Object,
toString()Ljava/lang/String; > v7 @26 exception:v8(line 20) {7=[t3]}
BB7
19   return                              (line 21)
BB8
```

# Sample 1 – Expected Output

```
BB0
BB1
2   v4 = new <Application,LTestCases/SampleTests>@2(line 13)
BB2
4   invokespecial < Application, LTestCases/SampleTests, <init>()V > v4
@6 exception:v5(line 13)
BB3
10   conditional branch(le, to iindex=14) v1,v6:#10(line 15) {1=[i]}
BB4
13   goto (from iindex= 13 to iindex = 16)   (line 17)
BB5
BB6
         v7 = phi  v3:#null,v4
17   v9 = invokevirtual <
Application, Ljava/lang/Object, toString()Ljava/lang/String; > v7 @26
exception:v8(line 20) {7=[t3]}
BB7
19   return                              (line 21)
BB8
```

```
BB0 -> BB1:
BB1 -> BB2: {(v4 ->{new2})}
BB2 -> BB3: {(v4 ->{new2})}
BB3 -> BB4: {(v4 ->{new2})}
BB4 -> BB5: {(v4 ->{new2})}
BB5 -> BB6: {(v4 ->{new2})}
BB6 -> BB7: {(v4 ->{new2}), (v7->{NULL, new2})}
BB7 -> BB8: {(v4 ->{new2}), (v7->{NULL, new2})}
```

## Eclipse

- You are encouraged to develop your tool within Eclipse.
- Within Eclipse, the arguments to PAVNullDerefAnalysis.java will be provided by the “Run Configuration” option in Eclipse (Run -> Run Configuration -> Arguments)

## Setting up the Project

- Follow the instructions in SetUp.pdf provided to you

## Other Important Information

## Other Information

- Do NOT import the com.ibm.wala.ipa.callgraph.propagation package.
  - Doing so will result in zero marks for the phase under evaluation, even if you do not use any class from the package
- Test Cases
  - A few provided right at the beginning
  - Evaluation to be done on more, previously undisclosed, test cases
- Do NOT modify any code region marked NO CHANGE REGION.
  - Example on next slide

## Example

```
/*
* Skeleton main method. Initialize the variables appropriately and call the necessary functions.
* START: NO CHANGE REGION
*/
String classpath, mainClass, analysisClass, analysisMethod;

classpath = args[1];
mainClass = args[2];
analysisClass = args[3];
analysisMethod = args[4];

PAVPointerAnalysis pAnalysis = new PAVPointerAnalysis(classpath, mainClass, analysisClass, analysisMethod);
pAnalysis.runAnalysis();
// END: NO CHANGE REGION
```

## Evaluation

- What we are looking for:
  - Your tool should not crash
  - The output should be sound
    - **Unsoundness**: *an edge OR node which should be in the points to set, is not present*
  - No unexpected imprecision
    - **Imprecision**: *an edge OR node which need not be in the points to set, is present*

- Scoring:
  - Each error has an associated penalty
  - Your score: TOTAL SCORE – sum(PENALTIES)

## Evaluation

- Demo of Phase 1:
  - Date: November 5th, 2018.
  - During demo: *run your tool on predisclosed (public) as well undisclosed (private) testcases*
- Credits will be divided between Phase I and Phase II
- No changes to the score of Phase I shall be entertained after the demo of Phase I

## Also,

- Your code will be carefully analyzed **with plagiarism checkers.**
- Copying will be dealt with severely.
- Both teammates need to participate. During the demo, we will be evaluating the responses of both members.
- Ideally, we would like to see the commits of both members. Nevertheless, this is not strictly enforced.

# References

- http://wala.sourceforge.net/javadocs/trunk/
- https://github.com/wala/WALA/wiki/Getting-Started

Thank you