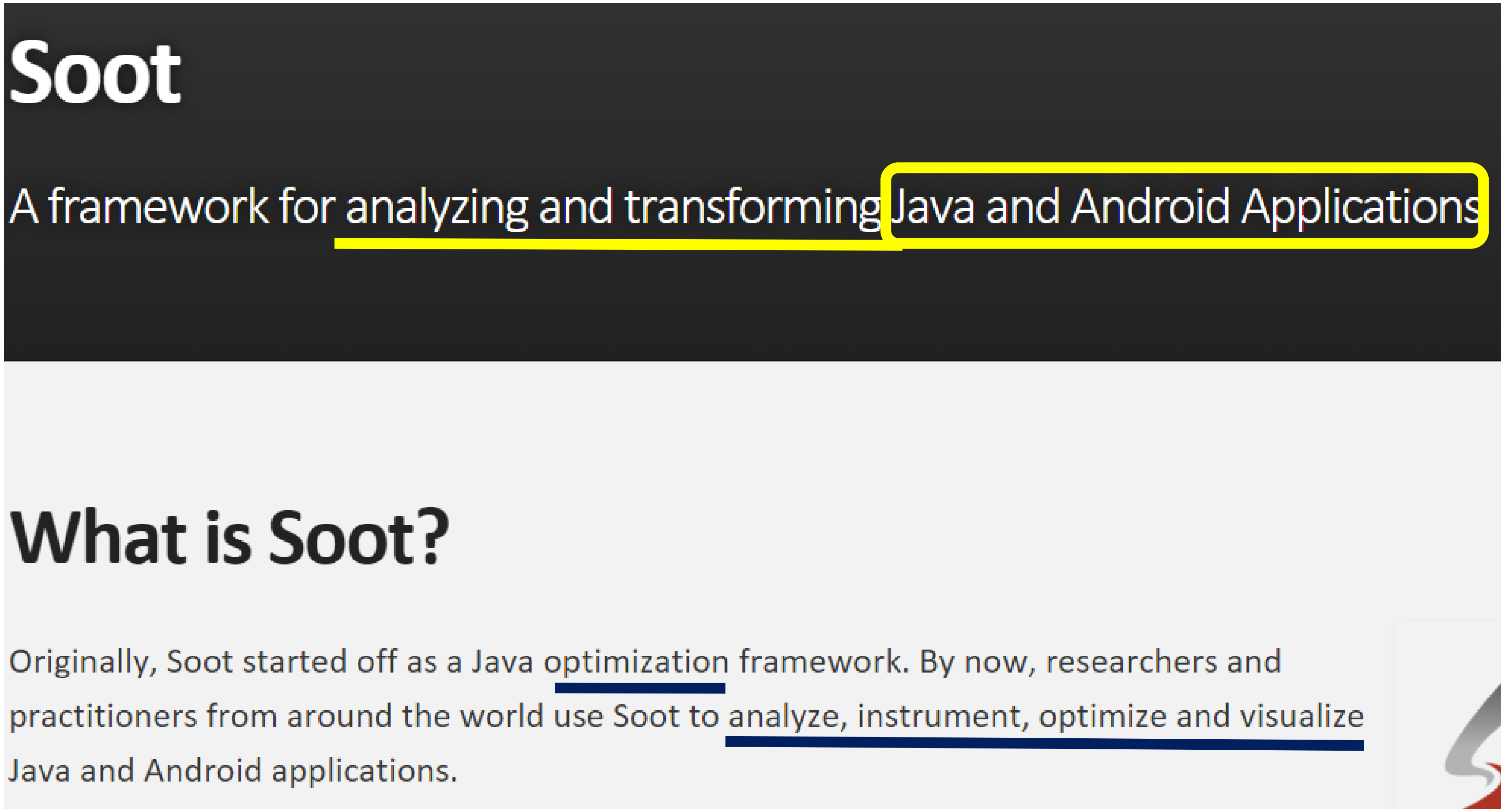
一个Java程序分析与变换框架 &过程间分析

唐浩
tanghao13@sei.pku.edu.cn

《软件分析技术》课程
2017.10.18

主页 http://sable.github.io/soot/



Soot

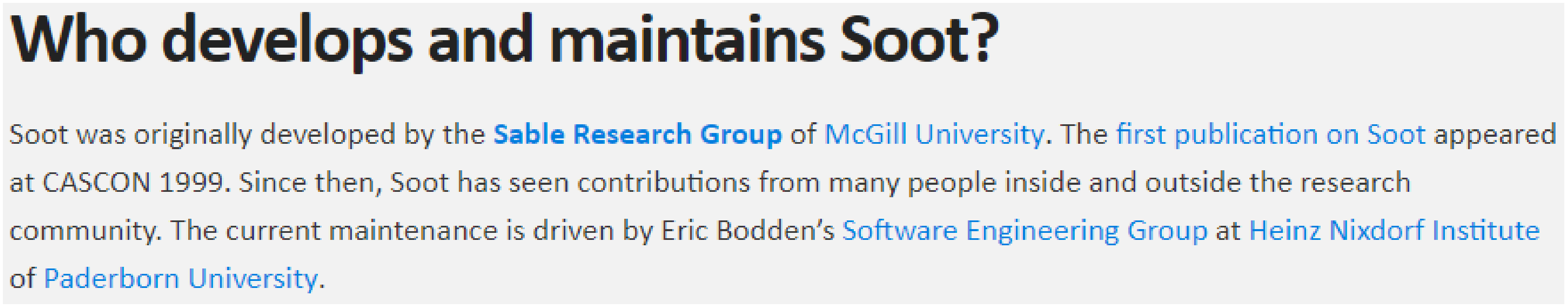
A framework for analyzing and transforming Java and Android Applications

What is Soot?

Originally, Soot started off as a Java optimization framework. By now, researchers and practitioners from around the world use Soot to analyze, instrument, optimize and visualize Java and Android applications.

主页 http://sable.github.io/soot/

- 历史



Who develops and maintains Soot?

Soot was originally developed by the **Sable Research Group** of McGill University. The first publication on Soot appeared at CASCON 1999. Since then, Soot has seen contributions from many people inside and outside the research community. The current maintenance is driven by Eric Bodden's Software Engineering Group at Heinz Nixdorf Institute of Paderborn University.

- 后续产品: FlowDroid …

Soot: Input & Output

- Input：Java源代码

- Output：程序分析的结果（例如活跃变量、指针指向集合）

Q: 分析Java源代码的第一步？

- 困难：直接分析源码文本，难以知悉代码结构
- 转为**中间代码**
  - 词法分析、句法分析、语义分析（、代码变换）
- 为什么要转成中间代码？
  - 保留源码信息（映射关系明确）
  - 方便机器理解（简单化、结构化）

面向程序分析的中间代码

- 直接利用Java中间代码Bytecode（字节码）
  - 太贴近机器码（为执行而设计）
  - 语句类型达199种
    - https://en.wikipedia.org/wiki/Java_bytecode
    - https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings
  - 基于栈的代码

面向程序分析的中间代码

- 直接利用Java中间代码Bytecode（字节码）
  - 基于栈的代码

```
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic       #84; // Field java/lang/System.out:Ljava/io/PrintStream;
34:  iload_1
35:  invokevirtual   #85; // Method java/io/PrintStream.println:(I)V
38:  iinc    1, 1
41:  goto    2
44:  return
```

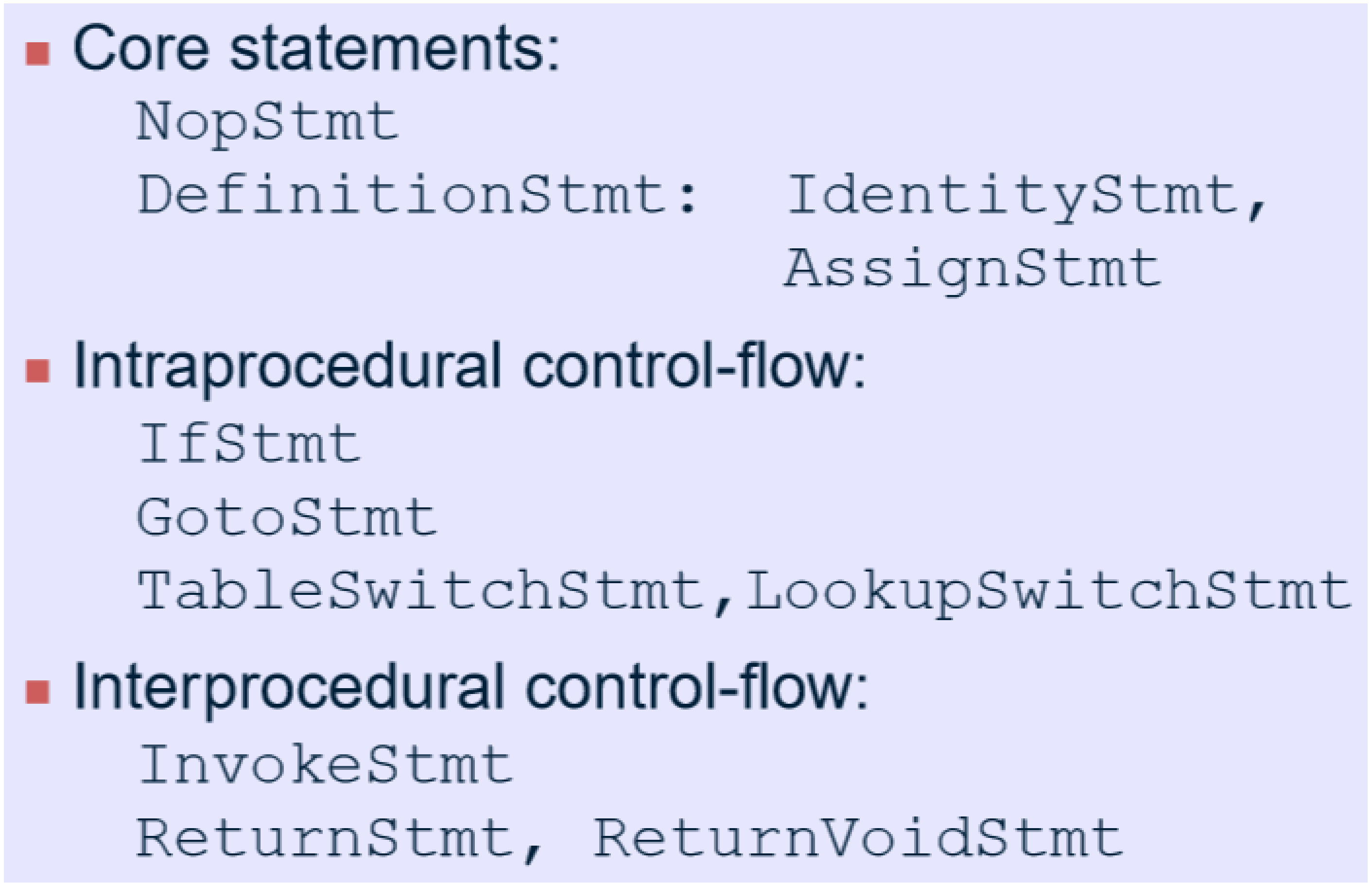
面向程序分析的中间代码

- Soot的中间代码——适合程序分析
  - Baf
  - **Jimple**
    - “Jimple is the principal representation in Soot. The Jimple representation is a typed, 3-address, statement based intermediate representation.”
    - 实际转换过程：source code -> bytecode -> Jimple
    - 15种语句
  - Shimple
  - Grimp
  - Dava

“A Survivor's Guide to Java Program Analysis with Soot”: http://www.brics.dk/SootGuide/

面向程序分析的中间代码

- Jimple

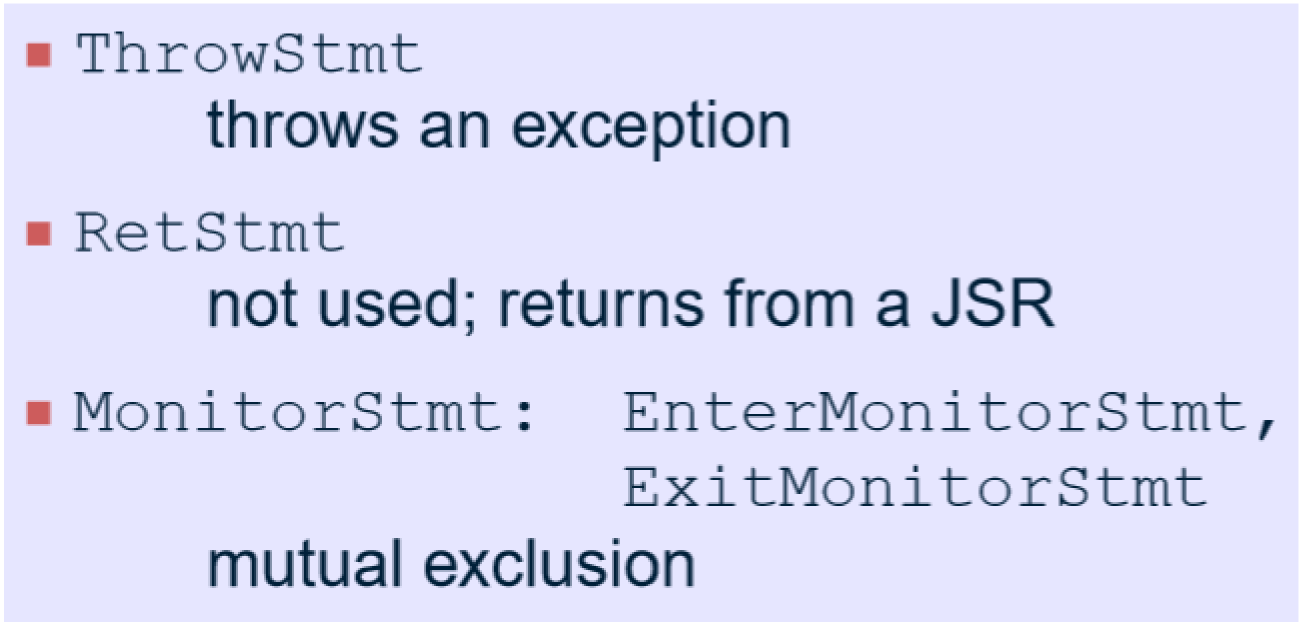


- Core statements:
```
NopStmt
DefinitionStmt:  IdentityStmt,
                 AssignStmt
```
- Intraprocedural control-flow:
```
IfStmt
GotoStmt
TableSwitchStmt,LookupSwitchStmt
```
- Interprocedural control-flow:
```
InvokeStmt
ReturnStmt, ReturnVoidStmt
```

http://www.iro.umontreal.ca/~dufour/cours/ift6315/docs/soot-tutorial.pdf

面向程序分析的中间代码

- Jimple



- `ThrowStmt`
  throws an exception
- `RetStmt`
  not used; returns from a JSR
- `MonitorStmt:   EnterMonitorStmt,`
  `                ExitMonitorStmt`
  mutual exclusion

http://www.iro.umontreal.ca/~dufour/cours/ift6315/docs/soot-tutorial.pdf

上机实践(1): Java→Jimple

- 命令行执行Soot
  - 注意设置-soot-class-path(-cp)
  - 输出为Jimple的选项：-f J
  - 命令行说明 Soot command-line options：https://ssebuild.cased.de/nightly/soot/doc/soot_options.htm
  - 扩展阅读：http://www.bodden.de/2008/08/21/soot-command-line/

- 编写Java程序执行Soot
  - 模拟命令行执行Soot
  - soot.Main.main(args);

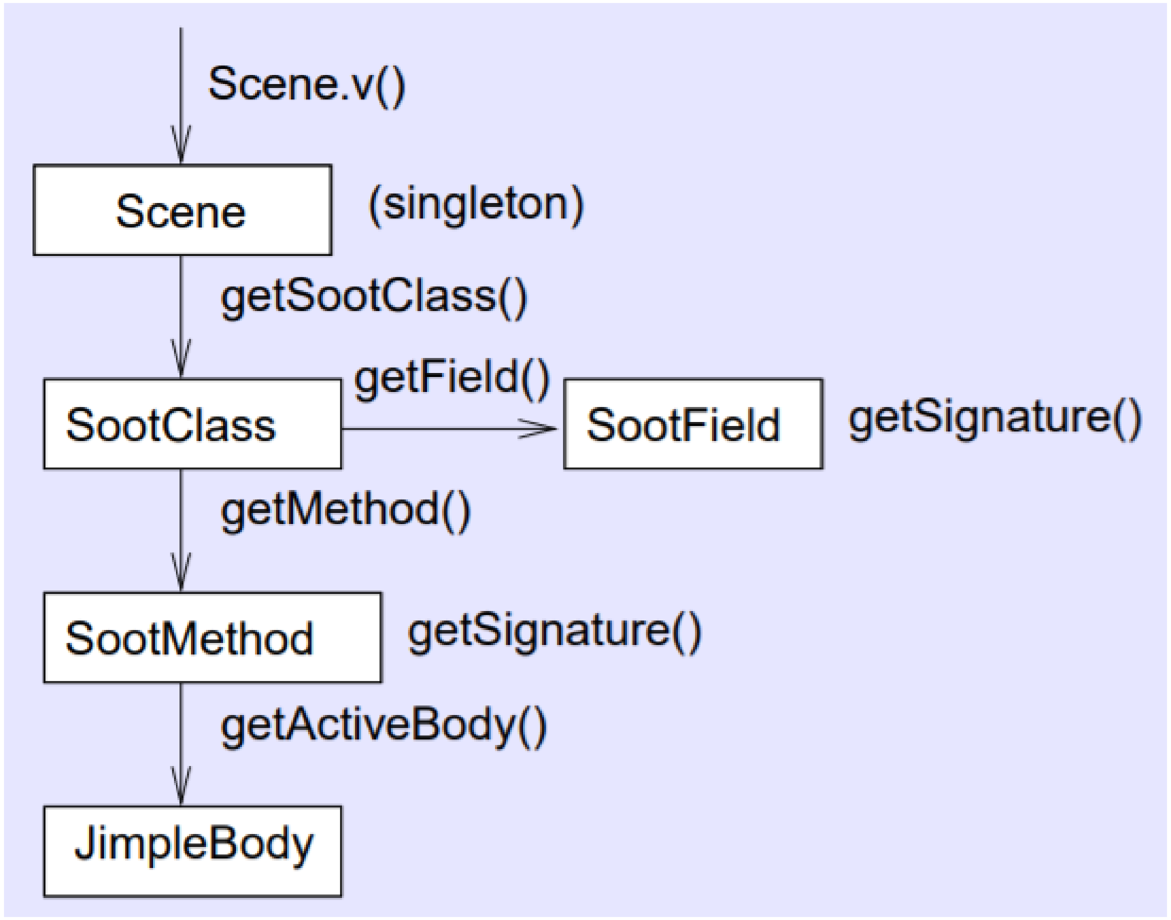
遍历程序结构

- 面向对象技术实现

- “环境”：Scene
- 类：SootClass
- 域：SootField
- 方法：SootMethod
- 函数体：Body / JimpleBody
- 语句：Unit

扩展阅读：https://github.com/Sable/soot/wiki/Fundamental-Soot-objects

遍历程序结构


Scene.v()
Scene
(singleton)
getSootClass()
SootClass
getField()
SootField
getSignature()
getMethod()
SootMethod
getSignature()
getActiveBody()
JimpleBody

Packs & Phases

- https://github.com/Sable/soot/wiki/Packs-and-phases-in-Soot
- Whole-program packs

```
public static void main(String[] args) {
  PackManager.v().getPack("wjtp").add(
      new Transform("wjtp.myTransform", new SceneTransformer() {
        protected void internalTransform(String phaseName,
            Map options) {
          System.err.println(Scene.v().getApplicationClasses());
        }
      }));
  soot.Main.main(args);
}
```

扩展阅读：http://www.bodden.de/2008/11/26/soot-packs/

Call Graph (whole program)
+ Control Flow Graph (each method)

- Scene.v().getCallGraph()

- Body body = xxMethod.getActiveBody();
- BriefUnitGraph g = new BriefUnitGraph(body);

上机实践(2)

- Call graph + Control flow graph
- 访问一个Class的所有Field和Method
- 遍历一个Method的所有语句(Unit)
- 求一个Class的子类和父类

指针分析

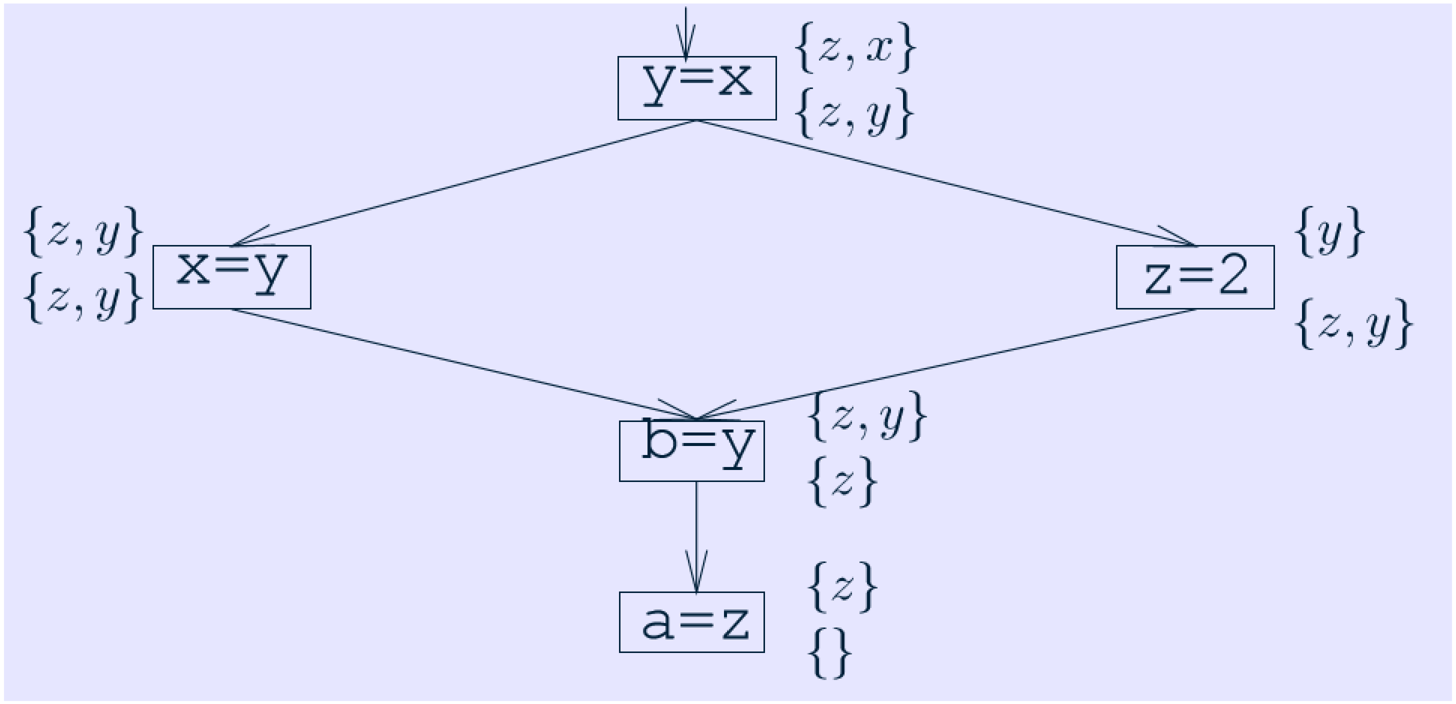
- getPointsToAnalysis…
- Spark
    - -p cg.spark enabled:true
- Paddle

上机实践(3)：指针分析

- 在MainCFA.java上做实验
- 输出PointsToSet内部信息

数据流分析

- 活跃变量分析



数据流分析

- 查阅文档
  - https://www.sable.mcgill.ca/soot/doc/soot/toolkits/scalar/AbstractFlowAnalysis.html
  - …

上机实践(4)：数据流分析

- 利用BackwardFlowAnalysis类完成活跃变量分析
- 打印每条语句前后的活跃变量集合

资料

- “A Survivor's Guide to Java Program Analysis with Soot”: http://www.brics.dk/SootGuide/
- 浅显易懂的教程：http://www.iro.umontreal.ca/~dufour/cours/ift6315/docs/soot-tutorial.pdf
- GitHub项目： https://github.com/Sable/soot/
  - **官方教程：https://github.com/Sable/soot/wiki/Tutorials**
  - **API： https://ssebuild.cased.de/nightly/soot/javadoc/**
- The Soot framework for Java program analysis: a retrospective
  - http://sable.github.io/soot/resources/lblh11soot.pdf

作业

- 访问一个Class的所有Field和Method
  - 自行编写被测程序的源码
    - 3-5个field、2-4个method
  - 提交内容
    - 被测程序的源码
    - 分析程序的源码
    - 执行结果的截图