Soot: a framework for analysis and optimization of Java

www.sable.mcgill.ca

Java .class files

- Contain fields, methods and attributes
- Fields: instance variables or class variables
- Methods: contain Java bytecode

// java source
int cc (int x, int y) {
    int z;
    z = x*y;
    return z;
}

// bytecode (javap –c)
Method int cc (int, int) {
    0 iload 1
    1 iload 2
    2 imul
    3 istore 3
    4 iload 3
    5 ireturn
}
### jimple files

- An Intermediate Representation

```java
// java source
int cc (int x, int y) {
    int z;
    z = x*y;
    return z;
}

// bytecode(javap -c)
Method int cc (int, int) {
    0 iload 1
    1 iload 2
    2 imul
    3 istore 3
    4 iload 3
    5 iload 3
    6 return }

// jimple(java soot.Main -f jimple)
int cc(int, int)
{  int i0, i1, i2;
   i0 := @parameter0: int;
   i1 := @parameter1: int;
   i2 = i0 * i1;
   return i2;
}
```

---

### Intermediate Representations

- Bytecode vs. 3-address code

**Bytecode:**
- Each instruction has implicit effect on stack
- No types for local variables
- > 200 kinds of insts

**Typed 3-address code:**
- Each stmt acts explicitly on named variables
- Types for each local variable
- Only 15 kinds of stmt

Do analysis on JIMPLE 3-address code IR.
Intermediate Representations

- Source vs. 3-address code

Source
- Irregular structure (somewhat)
- Complex statements and expressions

3-address code:
- More regular structure
- 15 kinds of stmts, simple expressions and statements

Analysis is simpler and more effective on JIMPLE 3-address code than source!

Overview of Soot

Class files

JIMPLIFY

ANALYSIS/ OPTIMIZATION

Optimized jimple

Grimp IR

Baf IR

Class files
Understanding Jimple

- Run soot: `java soot.Main -f jimple MyClass`

```java
public class A {
    public void m() {
        A r0;
        r0 := @this: A;
        specialinvoke r0.
        <java.lang.Object: void <init>()>();
        return;
    }

    public class A extends java.lang.Object {
        public void <init>() {
            A r0;
            r0 := @this: A;
            specialinvoke r0.
            <java.lang.Object: void <init>()>();
            return;
        }
    } ...
}
```

---

Understanding Jimple, cont.

```java
public class A {
    public void m() {
        A r0;
        r0 := @this: A;
        return;
    }

    public class A extends java.lang.Object {
        public void <init>() {
            A r0;
            r0 := @this: A;
            return;
        }
    } ...
}
```
Understanding Jimple, cont.

```java
public class A {  
    main(String[] args) {  
        A a = new A();  
        a.m();  
    }  
    public void m() {  
    }  
}

main(java.lang.String[]) {  
    java.lang.String[] r0;  
    A $r1, r2;  
    r0 := @parameter0: java.lang.String[];  
    $r1 = new A;  
    specialinvoke $r1.<A: void <init>>();  
    r2 = $r1;  
    virtualinvoke r2.<A: void m>();  
    return;  
}  
```

Resources for SOOT

- paul: /grad/cs515/soot222/soot-2.2.2/tutorial
Spark in the Context of SOOT

Spark Overview (with Pre-Built Call Graph)
Spark Overview (Call Graph Built On the fly)

Jimple Code → Native Method Simulator → Pointer Assignment Graph (PAG) Builder → Call Graph

Call Graph Updater

PAG

Points-to Set Propagator

Points-to Result

Points-to Set Propagator

- Four Statements
  - Allocation (a=new C())
  - Load (a=b.f)
  - Store (b.f=a)
  - Assignment (a=b)

- Type Filter
  - v points-to o => type(o) is compatible with (subtype of) type(v).
  - Type filter for Assignment and Load statements
Spark Options

- points-to set propagation option (soot.jimple.spark.solver.Propagator)
  - iter: a simple iterative algorithm
  - alias: alias edge based, requires small memory
  - worklist: work list based, fastest

- points-to set implementation
  - hash: java built-in hash set
  - bit: bit vector
  - hybrid: two choices depending on the set size
  - array
  - double: two sets for each points-to set, propagated and not yet propagated (by default two sets are both hybrid)

Command to Run Spark

- To run the program: java –Xmx512m soot.Main --app -p cg.spark on-fly-cg:true -w TargetJavaApplication
  - The Running Time may be long on paul!
  - You can try to set the option for on-fly-cg to false and see what will happen.
- -app : application mode, processing all possible reachable classes
- -w: whole program mode
- More Soot command line options please refer to
  http://www.sable.mcgill.ca/soot/tutorial/usage/
Resources for SPARK

- Spark Options:
  http://www.sable.mcgill.ca/soot/tutorial/phase/phase.html#SECTION00042000000000000000

- Master Thesis by Ondřej Lhoták
  “Spark: A flexible points-to analysis framework for Java”

- Source Code: soot-2.2.2/src/soot/jimple/spark
  SparkTransformer.java