Soot

http://www.sable.mcgill.ca/soot/

Acknowledgements

- Some of the graphics and code segments are taken from tutorials found on Soot homepage

Soot

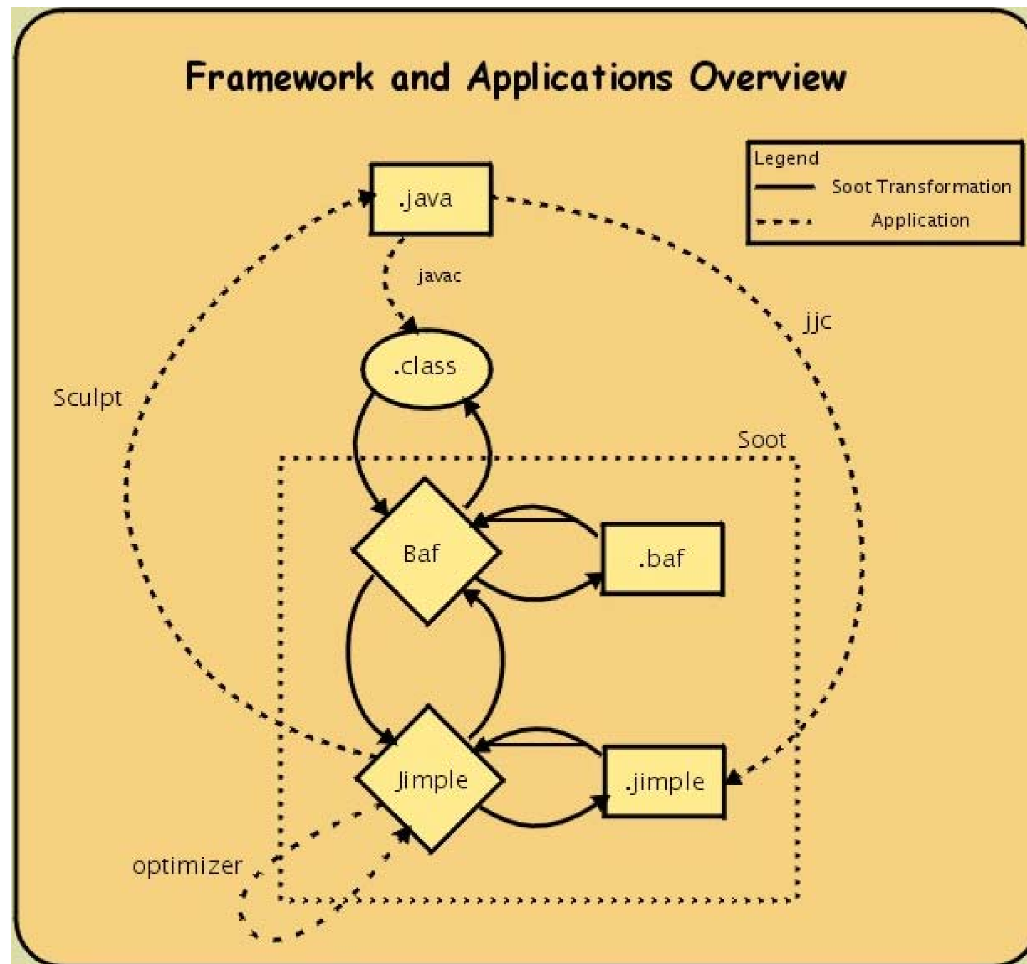
- A Java bytecode analysis and transformation framework
- Motivation: Java bytecode is hard to analyzed and optimized, need a better representation before we can do analyses.

Soot as an optimizer

- Soot does have built-in optimization modules (constant, copy propagation, dead code elimination, method inlining…)

- But optimization is not what Soot is about. It's a framework that supports/facilitates the addition of user-defined analyses

Framework Overview



Soot IR's

- Several different IR's:
  - Baf (bytecode like)
  - Jimple (3-address code)
  - Shimple (SSA Jimple)
  - Grimp (aggregated Jimple)

Sample code

```
public void compute() {
    int x = 1;
    while (opaque) {
        if (x!=1)
            x = 2;
    }
}
```

Baf

- Similar to Java Bytecode
- But:
  - Eliminate the constant pool table
  - Consolidate all different arithmetic operators (only one simple add instruction)
- Useful for bytecode-based analyses

Baf code

```
public void compute()
{
  word r0, b0;

  r0 := @this: Original;
  push 1;
  store.b b0;

label0:
   load.r r0;
   fieldget <Original:
 boolean opaque>;
   ifeq label1;
```

```
load.b b0;
        push 1;
        ifcmpeq.b label0;

        push 2;
        store.b b0;
        goto label0;

     label1:
        return;
    }
```

Jimple

- The main IR for Soot
- Characteristics:
  - Typed
  - 3-address: complex statements are “linearized”
  - Statement-based: 15 types of statements
    - Core Statements: Nop, Identity, Assign
    - Intraprocedural Control Flow: If, Goto, TableSwitch, LookupSwitch…
    - Interprocedural Control Flow: Invoke, Return, ReturnVoid
    - Monitor: Enter, Exit
    - Some other special statements…

Jimple code

```
public void compute()
    {
        Original this;
        byte x;
        boolean $z0;

        this := @this: Original;
        x = 1;

     label0:
        $z0 = this.<Original: boolean opaque>;
        if $z0 == 0 goto label1;

        if x == 1 goto label0;

        x = 2;
        goto label0;

     label1:
        return;
    }
```

Shimple

- SSA form of Jimple
- Several optimizations run better on SSA code

Shimple code

```
 public void compute()
    {
        Original this;
        byte x, x_1, x_2;
        boolean $z0;

        this := @this: Original;
(0)     x = 1;

     label0:
        x_1 = Phi(x #0, x_1 #1, x_2 #2);
        $z0 = this.<Original: boolean opaque>;
        if $z0 == 0 goto label1;

(1)     if x_1 == 1 goto label0;

        x_2 = 2;
(2)     goto label0;

     label1:
       return;
    }
```

Shimple vs Jimple

- Conditional Constant Propagation is only implemented to run on SSA code (gcc's CCP is also SSA-based)

- It's the only built-in advantage right now…

Optimized Jimple code

```
public void compute()
    {
        Original this;
        byte x;
        boolean $z0;

        this := @this: Original;
        x = 1;

     label0:
        $z0 = this.<Original: boolean opaque>;
        if $z0 == 0 goto label1;

        if x == 1 goto label0;

        x = 2;
        goto label0;

     label1:
        return;
    }
```

Optimized Shimple code

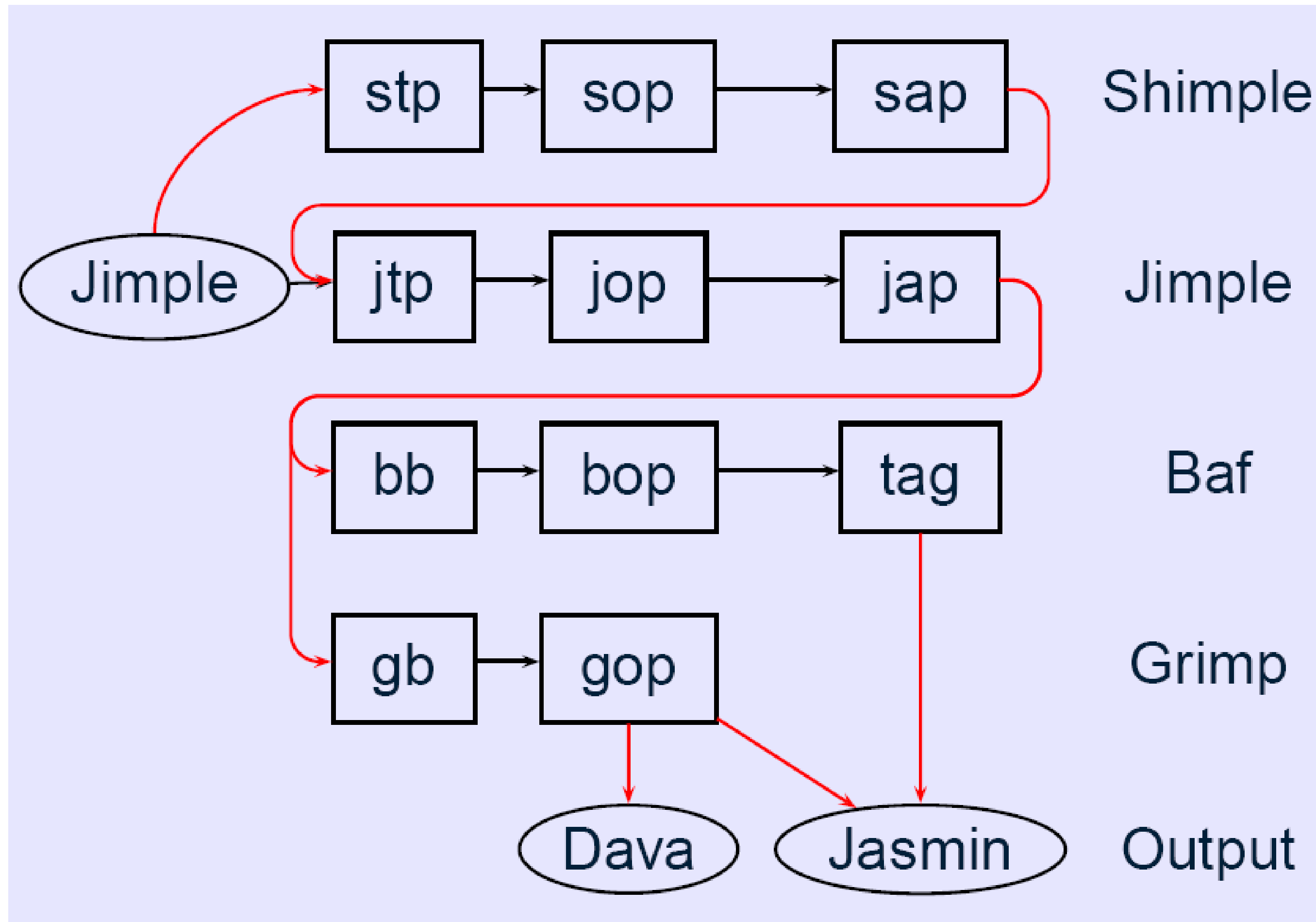
```
public void compute()
    {
        Original this;
        boolean $z0;

        this := @this: Original;

     label0:
        $z0 = this.<Original: boolean opaque>;
        if $z0 == 0 goto label1;

        goto label0;

    label1:
       return;
    }
```

Soot Transformation

$$w^{?}(j|s|b|g)(b|t|o|a)p$$

- w $\Rightarrow$ Whole-program phase
- j, s, b, g $\Rightarrow$ Jimple, Shimple, Baf, Grimp
- b, t, o, a $\Rightarrow$
  - (b) Body creation
  - (t) User-defined transformations
  - (o) Optimizations with -O option
  - (a) Attribute generation

Soot Transformation



Basic Elements of Soot

- Scene: The complete “universe”
- Class: A single loaded class
- Body: Method body
- Unit: line of code (not necessarily line of source code)
- Box: Various parts of a unit

A Small Demo

- How all the elements work together
- Demo: Swapping two operands of addition

Demo

- First need to add user-defined “pack” into the Transformation framework

```
package soot;

public class MyMain {
   public static void main (String [] args) {
        PackManager.v().getPack("jtp").add (
        new Transform("jtp.MyTransform",MyTransform.v()));
        Main.main(args);
   }
}
```

Demo

```
package soot;

import java.util.*;
import soot.jimple.*;

public class MyTransform extends BodyTransformer {
    private static MyTransform instance = new MyTransform();
    private MyTransform() {}
    public static MyTransform v() {return instance;}

    protected void internalTransform(Body b, String phaseName, Map options) {
          Iterator unitIt = b.getUnits().iterator();
          while (unitIt.hasNext()) {
                    Unit ut = (Unit) unitIt.next();
                    Iterator valueBoxIt = ut.getUseBoxes().iterator();
                    while(valueBoxIt.hasNext()) {
                              ValueBox vb = (ValueBox) valueBoxIt.next();
                              Value v = vb.getValue();
                              if (v instanceof AddExpr) {
                                        Value op1 = ((AddExpr)v).getOp1();
                                        Value op2 = ((AddExpr)v).getOp2();
                                        ((AddExpr)v).setOp1(op2);
                                        ((AddExpr)v).setOp2(op1);
                              }
                    }

          }
    }
}
```

Flow Analysis in Soot

- Provides 3 different flow analyses: `ForwardFlow`, `BackwardFlow`, and `ForwardBranchFlow`
- Soot has two built-in flow analyses: `Liveness` and `NullPointer`

Flow Analysis Checklist

- Forward or backward?
- May or must?
- Transfer function?
- Initial State?

Example

- From Soot's Survival Guide
- An analysis for Very Busy Expression
  - An expression is busy at a given point if it's evaluated on all path taken at that point.
  - Useful for code hoisting

VBE Analysis in Soot

- Forward/Backward

  VBE is a backward analysis

```
class VBEAnalysis extends BackwardFlowAnalysis {
  public VBEAnalysis(DirectedGraph g) {
          super(g);
          doAnalysis();
  }
  …
  …
}
```

VBE Analysis in Soot

- May / Must

  VBE is a must-analysis

```
protected void merge (Object in1, Object in2, Object out) {
  FlowSet inSet1 = (FlowSet) in1, inSet2 = (FlowSet) in2,
          outSet = (FlowSet) out;
  inSet1.intersection (inSet2, outSet)
}
protected void copy (Object source, Object dest) {
  FlowSet srcSet = (FlowSet) source, destSet = (FlowSet)dest;
  srcSet.copy (destSet)
}
```

VBEAnalysis

- Transfer function

VBEIn(b) = GEN(b) U (VBEOut(b) – Kill(b))

```
protected void flowThrough(Object in, Object node, Object out) {
        FlowSet inSet = (FlowSet)source,
        outSet = (FlowSet)dest;
        Unit u = (Unit)node;
        kill(inSet, u, outSet);
        gen(outSet, u);
}
```

Initial State

```
protected Object entryInitialFlow() {
  return new ValueArraySparseSet();
}
protected Object newInitialFlow() {
  return new ValueArraySparseSet();
}
```

Summary

- Soot is a useful framework for analyzing programs
- Easy integration with user-defined modules
- Eclipse plugin is a plus
- Documentation: Fair