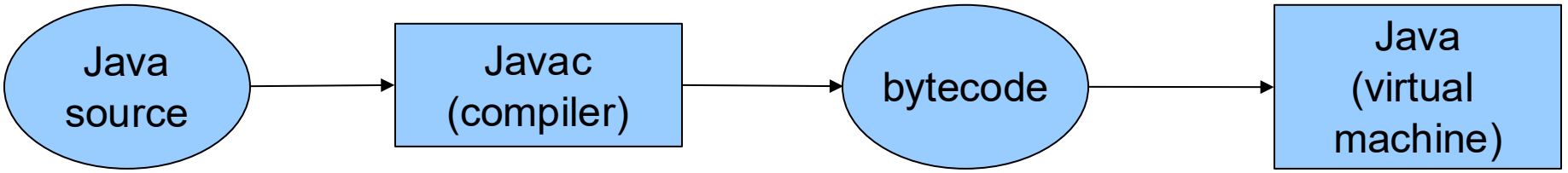
Extendiendo SOOT

La máquina virtual Java

- El compilador Java traduce un programa Java en el lenguaje de entrada de la Java Virtual Machine (JVM)
- El bytecode Java es una suerte de lenguaje "assembler" para la JVM



El framework Soot

- Conjunto de APIs de Java para operar con bytecode Java
  - optimización
  - anotación
- Creado por el Sable Research Group (http://www.sable.mcgill.ca/)
- Web: http://www.sable.mcgill.ca/soot/

Representaciones Intermedias

Jimple: ppal representación de Soot

Grimp: Jimple con expresiones

Shimple: Jimple con SSA

Baf: bytecode "humanizado"

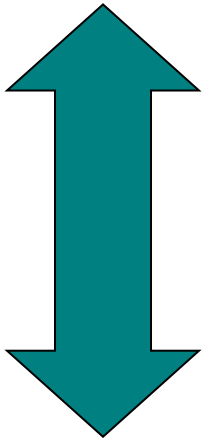
Representaciones Intermedias

Java

Grimple (alto)

Jimple/Shimple (medio)

Baf (bajo)



Bytecode

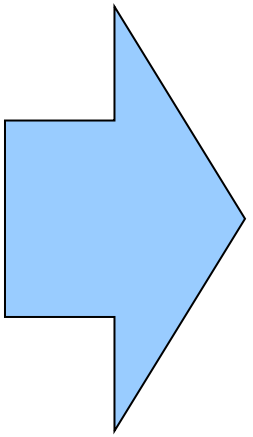
Jimple

- Puede crearse usando:
  - programa Java
  - bytecode Java
- Características :
  - 3 referencias: Todas las expresiones usan a lo sumo 3 direcciones (3 address code)
  - No-estructurado: while's, if's, for's son reemplazados por GOTO's
  - Tipado: las variables locales son tipadas

Jimple: Ejemplo

- Programa Java original
- Representación Jimple

```
if (x+y!=z)
  return;
else
  System.out.println(“foo”);
```



```
  t = x+y;
  if (t==z) goto label10;
  return;

label10:
  ref = System.out;
  ref.println(“foo”);
```

Dataflow framework

1. Decidir dirección del flujo: ¿backward o forward?
2. Decidir aproximación: ¿may o must?
3. Realizar la transformación del flujo: ej: ¿cómo deben tratarse las asignaciones?
4. Decidir estados iniciales y el flujo inicial del entry/exit node (dependiendo del tipo de análisis)

1.Dirección del flujo

- Soot posee 3 implementaciones de análisis :
  - ForwardFlowAnalysis
  - BackwardFlowAnalysis
  - ForwardBranchedFlowAnalysis
- La salida es un Map<Nodo,<IN set, OUT set>>

```
public class MyFwdAnalysis extends ForwardFlowAnalysis<Unit,FlowSet> {
  public MyFwdAnalysis(DirectedGraph<Unit> graph) {
    super(graph);
    doAnalysis();
  }}
```

2. Aproximación

- Implementar los métodos “merge” y “copy”

```
protected void merge(FlowSet intSet1,
                     FlowSet inSet2,
                     FlowSet outSet) {
   inSet1.intersection(inSet2, outSet);
}

protected void copy(FlowSet srcSet,
                    FlowSet destSet) {
   srcSet.copy(destSet);
}
```

3. Transformación del flujo

- Implementar el método flowTrough

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

- Implementa la transfer function.
- Los métodos kill y gen (son opcionales)
- Como pasa el **in** al **out** para esa instrucción (en **node**)

4. Flujos iniciales

- Hay que definir el contenido inicial del entry/exit point, y de los otros nodos del CFG

```
protected FlowSet entryInitialFlow() {
  return new FlowSet();
}
protected FlowSet newInitialFlow() {
  return new FlowSet();
}
```

Hay distintos tipos de FlowSet

- ArraySparseSet, ArrayPackedSet, ToppedSet
- Definidos por el usuario (ej: Var->Sign)

Ejecutando el análisis

```
Scene.v().setSootClassPath("...")
SootClass c = Scene.v().loadClassAndSupport("ar.edu.soot.MyClass");
c.setApplicationClass();
SootMethod m = c.getMethodByName("myMethod");
Scene.v().loadNecessaryClasses();
Body b = m.retrieveActiveBody();
UnitGraph g = new ExceptionalUnitGraph(b);
MyFwdAnalysis an = new MyFwdAnalysis(g);
for (Unit unit : g) {
  FlowSet in = an.getFlowBefore(unit);
  FlowSet out = an.getFlowAfter(unit);
}
```

Levantamos la clase y leemos el metodo

Creamos el CFG

Invocamos a nuestro dataflow sobre el CFG

Miramos lo que dio el punto fijo para cada nodo del CFG