Using the Jimple Parser

Patrice Pominville
patrice@sable.mcgill.camailto:patrice@sable.mcgill.ca

March 24, 2000

1 Introduction

At the heart of the Soot bytecode optimization framework is the Jimple Internal Representation. This document describes Soot's ability to read in textual Jimple files.

We have already seen Soot's ability to load Java `.class` files. Soot can output the Jimple representation of classes in a textual format (`.jimple` files) and reread this textual format back into Soot.

The ability to parsing textual `.jimple` files is quite handy. For example, a user could output classfiles as `.jimple` files, tweak these by hand, and then use Soot to parse them and output tweaked bytecode. Soot thus provides a high-level facility for editing class files. Another potential application of the `.jimple` files is to cache optimized classfiles for future analysis. Soot can, at a later time, read these files back in, and will not need to reoptimize the original classfile. This leads to faster compilation times. Finally, a user, or a software tool, can write a class in the Jimple textual format, use Soot to parse the Jimple, and thus create bytecode.

trade off: precision and recall

The Jimple format is self-explanatory. The SableCC grammar for textual Jimple can be found in the source code distribution of Soot; it is called `jimple.scc`. The following is an fragment of Jimple code:

```
{
    java.lang.String[] r0;
    long l0, $l1, $l2, $l3, $l4, $l5, $l6, $l7;
    int i0, $i1, $i2, i3;
    java.io.PrintStream $r1, $r2, $r3;
```

```
      r0 := @parameter0: java.lang.String[];
      i0 = 0;
      $i1 = 33 % 33;
      $i2 = $i1 + 2;
      if $i2 != 43 goto label0;

      i3 = 9;
      goto label1;

   label0:
      i3 = 4;
```

2 Producing `.jimple` Files

The default output format for Soot is classfiles. However, Soot can also produce output in other formats, i.e. textual Jimple. To output `.jimple` files, invoke Soot using the `-f jimple` command line option. For example,

```
java soot.Main -f jimple --app test
```

will output test and dependent classes as `.jimple` files.

3 Parsing `.jimple` Files

Out of the box, Soot will first look for a classfile to find the definition of a given class; only if this fails will it look for a `.jimple` definition. The command line option `--src-prec jimple` (source precedence) instructs Soot to preferentially resolve the class from a `.jimple` file, falling back on `.class` files. When resolving, Soot will look for `.jimple` files using the `soot-classpath` in the same manner as it does to find `.class` files. For example, to build on the previous example, the command

```
java soot.Main --src-prec jimple --app test
```

will read the class `test` from the file `test.jimple`, and its related classes from the other `.jimple` files produced in the previous step.

For more details on the `-f` and `--src-prec` options the user can refer to the Soot command line documentation../usage.

History

- March 24, 2000: Initial version.
- May 31, 2003: Updated for Soot 2.0.