Static Analyser Design

Michael Hope ¡michaelh@juju.net.nz¿

May 11, 2002

Abstract

This document describes the Static Analyser in Jaune, a ahead of time compiler for Java class files. The purpose of the Static Analyser is to optimise the output for code size and speed by computing what methods and classes can be created and by computing which virtual calls can be made statically instead. As a side effect, the Static Analyser also provides the support methods required for Jaune such as absolute method numbers.

This document is incomplete. The static analyser was deemed too complicated for a 1.0 release.

1 Rules

- For each class that can be instantiated, the static initialiser needs to be run.
- For each class that can be instantiated, the class initialiser may run.
- Each method that is invoked by another referenced method may run.
- Interface methods are hard. For each interface method that is invoked by another referenced method, scan the list of live classes. If a live class implements that method, the appropriate method may run.
- For each referenced method, if that method is overridden by a child class and the class can be instantiated then the overriding method may run.

It is hard to detect and inhibit the class field initialisation code for fields that are initialised but not used.

2 Algorithim

The interface and overrides rules above require the interface implementors list and overriders list to be computed ahead of time or the algorithm to be run iteratively. Choose a tree based scan, so we need to compute the lists first.

2.1 Setup

Compute the implementors list. The implementors list is the list of all classes that implement a given interface.

- Scan each loaded class.
- For each interface this class implements, add this class to the list of implementors for that interface.

Compute the overriders list. The overriders list is the list of all methods that override a given method.

- Scan each loaded class.
- Scan each method in this class.
- Scan up the parent class list for this method until the first definition of this method is found.
- Add this method to the original methods list of overriders.

2.2 Scan

- Starting at the root set of methods, scan each method according to the Method scan rules.

2.3 Method scan

- If this method has already been scanned, abort.
- Scan the declaring class according to the Class scan rules.
- Scan each opcode of this method.
- If this opcode is a 'new', scan the created class with the Class scan rules. Scan the initialiser of the created class using the Method scan rules.
- If this opcode is an Invoke Virtual, scan according to the Invoke Virtual rules.
- If this opcode is an Invoke Static, scan according to the Invoke Static rules.
- If this opcode is an Invoke Interface, scan according to the Invoke Interface rules.

2.4 Class scan

This set of rules describes how to scan a new class.

- If this class has already been scanned, abort.
- Mark this class as referenced.
- Scan the class initialiser using the Method scan rules.

2.5 Invoke Virtual

This set of rules describes how to scan a normal virtual method that is invoked.

- Scan this method and every method that overrides it using the Method scan rules.

2.6 Invoke Static

- Scan the declaring class according to the Class scan rules.
- Scan this method using the Method scan rules[1].

2.7 Invoke Interface

- If the invoked method has already been scanned, abort.
- Record this in the list of to be processed invokes.
- Note that this has to be iterative, as you can't tell which implementations of this method can be run until you know which classes may be created, which you can't tell until you know which methods are invoked.

[1] A static method can be obscured but cannot be overridden.