Optimizations using Variable Type Analysis

Felix Kwok (wkwok@sable.mcgill.ca)

September 5, 2000

This note explains how to apply Variable Type Analysis to whole-program optimizations. The user should first be familiar with material in both Soot command-line options and Phase options.

1 What is Variable Type Analysis?

Variable Type Analysis (VTA) is a linear algorithm that determines statically a set of possible runtime types of each variable in the program. This analysis is useful for virtual method call elimination: if VTA discovers that the receiver of a virtual invoke site can only have one possible type, we can use this information to inline the target method or to bind the method statically, so that virtual method lookup is no longer needed.

For more information on VTA, see the Sable technical report 1999-4.

2 Running whole-program optimizations

Soot provides tools for whole-program optimizations in the `wjop` pack, some of which uses VTA. To use these tools, one must run `soot` in whole-program mode and must have turned on optimization. This is accomplished by the command-line options `--app` and `-W`. Also, since VTA needs to analyze the bodies of the java library code, one must include the command-line option `-a` or `--analyze-context` to tell Soot to obtain Jimple code for library methods.

3 The `wjop` Pack

The `wjop` pack contains two transformers, `StaticMethodBinder` and `StaticInliner`. Only one transformer should be applied for each execution. By default, `StaticMethodBinder` is disabled and `StaticInliner` enabled. This can be changed by setting the `disabled` option for each transformer.

`StaticInliner` (phase `wjop.si`) does the following:

1. finds call sites which are monomorphic;
2. checks whether the call sites can be safely inlined. The inlining criteria are listed in Vijay Sundaresan's Master's thesis;
3. if the call site is safe to inline, inlines the body of the target into that of the caller.

`StaticMethodBinder` (phase `wjop.smb`) does the following:

1. finds call sites which are monomorphic (i.e. has only one target);
2. creates an new static method which has a body identical to the target, but whose first parameter is the object that used to be the receiver;
3. redirects the original call site to the newly-created static method.

By default, `StaticInliner` and `StaticMethodBinder` uses Class Hierarchy Analysis (CHA) to find monomorphic call sites. Changing the `VTA-passes` option can cause them to use VTA once or several times.

4 Including dynamically-loaded classes

If the program to be optimized loads classes dynamically using the `newInstance` method, Soot will be unable to tell statically which classes need to be resolved. In this case, the user will need to tell Soot explicitly which classes are loaded. This can be done using one of the following command-line options:

1. `--dynamic-path` lets the user specify paths under which all classes are considered potentially dynamic. This option works provided the class files do not belong to any packages (i.e. the fully quantified name is the same as the class name). For example, this will work for a stand-alone class file, but will not work for classes like `sun.security.provider.Provider`.

2. `--dynamic-packages` lets the user specify packages, separated by commas, for which all class files belonging to this package or any subpackage thereof are considered potentially dynamic. For instance, saying

   ```
   --dynamic-packages sun.security.provider
   ```

   will mark a class like `sun.security.provider.Provider` as potentially dynamic.

We currently do not provide a way for the user to specify individual classes as potentially dynamic, but we hope to do that in the future. Note: *The user must specify all potentially dynamic classes using one (or both) of the above, or the results of VTA may be incorrect.*

5 Examples

- ```
  java -mx300m soot.Main --app -W -a -p wjop.smb disabled:false -p wjop.si disabled
  -p wjop.smb VTA-passes:2 spec.benchmarks._201_compress.Main
  ```

  This command runs `StaticMethodBinder` instead of `StaticInliner`. It also asks Soot to apply VTA twice on the analysis. It analyzes library classes, but does not include any dynamic packages. The `-mx300m` switch is present so that the virtual machine is allowed to use more memory (300 Mb) than the default value (since whole-program analysis usually uses a lot of memory). Note that the switch for allowing more memory usage may be different depending on the virtual machine used.

- ```
  java -mx500m soot.Main --app -W -a --dynamic-packages
  java.text.resources,spec.benchmarks._213_javac SpecApplication
  ```

  This command runs `StaticInliner` with no VTA. It uses CHA to find monomorphic sites. It analyzes library classes, and it includes all classes in the packages `java.text.resources`, `spec.benchmarks._213_javac`, or any of their subpackages, as potentially dynamic classes. It allows the virtual machine to use 500 Mb of memory.

History

- September 5, 2000: Initial version.