Artifact Overview for Paper #35 of OOPSLA 2014

1 Introduction

This document is an overview of the artifact for paper #35 of OOPSLA 2014. The main components contained in the artifact package are:

1. VarJ: A command-line software tool that refactors Java source code. As described in the paper, VarJ adds wildcard annotations to parametric types or instantiations of generics that occur in the given source files.
2. Six large Java generic libraries that VarJ was applied to for evaluating the potential benefit of the tool. These libraries are included in the artifact to allow reproducing the experiments from the paper.
3. Unix shell scripts for running basic usage of the refactoring tool and for running the experiments from the paper.

This overview follows the instructions from [2] and consists of the two parts requested by the instructions:

1. The *Getting Started Guide* is in Section 2.
2. The *Step by Step Instructions* are in Section 3.

2 Getting Started Guide

The VarJ refactoring tool was compiled to the Java jar file `VarJ/src/VarJFrontend/VarJ.jar` using Oracle's Java compiler, `javac` version `1.6.0_65`. This jar file was included in the artifact package, so that a user does not have to build it. `VarJ.jar` should be executed from the command line and requires a Java 1.6+ runtime environment. A Java runtime is usually already installed, but it can be downloaded from [3]. No other software is required; VarJ should be able to run on the machine after a Java runtime is installed.

Since most of the files discussed in this document are within the directory `VarJ/src/VarJFrontend`, the following file path convention will be used for the remainder of this document: If a file path that does not start with `VarJ`, then it implicitly starts with the path `VarJ/src/VarJFrontend`.

The script `test.sh` runs basic usage of the refactoring tool. It is designed to be executed only in the directory that contains it (`VarJ/src/VarJFrontend`). Executing that script should run basic usage of applications within `VarJ.jar`. These test analyze Java code in the directory `unittests` and generate directories, `rewrittenSources_sig` and `rewrittenSources_bodies`, that will contain refactored code. Text files `rewriteinfo_sig.txt` and `rewriteinfo_bodies.txt` should also be generated; they will list the rewrites performed in the refactoring of the code in `unittests`.

2.1 Dependencies

This subsection is provided only to help with troubleshooting in case of difficulties with running `VarJ.jar`. Executing the helper scripts does not require the user to be aware of the binaries that `VarJ.jar` depends on. `VarJ.jar` contains class files that were compiled using the Scala compiler, `scalac` version `2.9.3`. `VarJ.jar` does *not* require Scala to be installed, but it does depend on the jar file `scala-library.jar`, which can be found in a Scala 2.9 installation. Using this artifact package does not require Scala to be installed because this package includes the file `VarJ/src/3rd_party_libs/scala-2.9.3/lib/scala-library.jar`. The helper scripts in this artifact (including `test.sh`) use that jar file to run `VarJ.jar`. If Scala is installed on the user's machine, he or she can execute `VarJ.jar` simply with the command '`scala VarJ.jar`'.

3 Step by Step Instructions

This section explains how to reproduce the experiments from the paper and explains how to run the refactoring tool and other applications within `VarJ.jar` for other purposes.

3.1 Reproducing Experiments from the paper

Helper scripts for running the experiments from the paper are in the directory `VarJ/src/VarJFrontend/scripts`. These scripts should be executed from the parent directory `VarJ/src/VarJFrontend`.

Executing `scripts/compare_sig_and_bodies.sh` will recompute the statistics shown in the tables from the paper. This script may take a few minutes to complete because a large amount of code is analyzed. It will also print out statistics to JSON [1] files and print LaTeX code to the file `testtex/table.tex` to allow regenerating the tables from the paper. To generate a PDF of the generated LaTeX code, just execute '`make`' from the directory `testtex`.

Descriptions of the other scripts are provided below:

1. `scripts/rewrite_compile_*.sh`: These scripts generate a refactored version of one of the analyzed libraries and compile the refactored version of the code. For example, `scripts/rewrite_compile_apache.sh` refactors the Apache commons-collection library contained in the folder `VarJ/analyzed_libs/collections-generic-4.01`. The refactored source code for this library will reside in the folder `rewrittenSources/analyzed_libs/collections-generic-4.01`. Class files that result from compiling the generated code will be in a directory named `build`.

The class files resulting from the refactored version of the Apache commons-collection library, for example, will be in the directory `rewrittenSources/analyzed_libs/collections-generic-4.01/src/java/build`. The script `scripts/rewrite_compile_all.sh` will rewrite and compile the refactored version of the code for all six of the libraries from the experiments. The scripts `scripts/rewrite_compile_*bodies.sh` will refactor the libraries using the method body analysis described in the paper.

2. `scripts/analyze_*.sh`: These scripts do not refactor code. They only perform the variance analysis (inferring definition-site variance), compute statistics from the analysis, and print out statistics to the file `testtex/table.tex`. The script `scripts/analyze_all.sh` will perform these steps for all six of the libraries from the experiments. The scripts `scripts/analyze_*bodies.sh` will perform the variance analysis using the method body analysis described in the paper.

3.2 Running `VarJ` in General

VarJ can be executed without using the helper scripts provided in the artifact package. Multiple frontends/classes can be executed using `VarJ.jar`. Recall that `VarJ.jar` can be executed by (1) using the Scala code runner (using the command '`scala VarJ.jar`') if Scala is installed or (2) just using Java runtime if the Scala 2.9 library jar file (`scala-library.jar`) is in your classpath [5]. Jar files can be executed with the `java` command line program [4] by passing `VarJ.jar` as the value for the option `-jar`.

Executing `VarJ.jar` will execute the class `ui.RewriteAllSources`, which takes in source files and will try to rewrite (add wildcards) to as many parametric types as it safely can. The input sources files will not be modified; refactored code is generated in the directory specified by the user.

Other frontend classes (classes with a main method) are available in `VarJ.jar`. Frontend classes in `VarJ.jar` can be executed by just adding `VarJ.jar` to your classpath; in this case, the `-jar` option should not be passed to `java`. Passing each frontend class the help options '`-h`' or '`--help`' will cause these applications to print out a help message listing the required and optional labeled arguments; required labeled arguments are annotated with '*' in the help message. Each frontend class requires a list of source files or directories containing source files to be given. Directories specified will be traversed recursively to find all of the Java source files that it contains. Frontend classes in `VarJ.jar` along with short descriptions are listed below. Example commands executing these classes can found in the script `test.sh`.

1. `ui.RewriteAllSources`: This class was described above. This application also prints out a 'modification specification' file, which is a text file that lists the rewrites performed in the refactored code. This application requires the user to specify the name of the modification specification file using either the options '`-m`' or '`--modfile`'. Refactored code is generated in the directory specified using the options '`-d`' or '`--outdir`'. To refactor code using the method body analysis from the paper, specify the option '`--bodies`'.

2. `ui.RewriteClasses`: This application is similar to `ui.RewriteAllSources` and takes in a superset of its options. Instead of rewriting all of the parametric types that can safely be updated, it will only rewrite the parametric types of declarations contained within the classes or interfaces specified by the user. These classes/interfaces are specified using the options '`-t`' or '`--type`'; this option can be repeated to specify multiple classes/interfaces.

3. `ui.RewriteSelected`: This application is similar to `ui.RewriteAllSources` and takes in a superset of its options. Instead of rewriting all of the parametric types that can safely be updated, it will only rewrite the parametric types of declarations specified in a JSON file. The input JSON file is specified using the option '`--declsfile`'. Two sample JSON files that were used with this program are `unittests/includesExcludes.json` and `unittests/paperexample.json`. In the JSON file, users can specify which declarations they want the types of to be rewritten and which declarations that they do not want to change. Declarations are fields, method arguments, method return types, or local variables. Declarations that are requested to be rewritten are given in a list labeled '`includes`'. Declarations that are excluded from rewritting are specified in a list labeled '`excludes`'.

4. `ui.FilesCLParser`: This application does not generate any code. It only prints out the inferred definition-site variances for each generic type parameter declared in the input source files.

5. `ui.LookupVar`: This application is similar to `ui.FilesCLParser` except that it only prints out inferred definition-site variances for type parameters in the generic specified using the options '`-g`' or '`--generic`'.

4 Questions

Please, email questions to `jaltidor@cs.umass.edu`.

References

[1] JSON: JavaScript Object Notation. `http://www.json.org/`.

[2] OOPSLA. OOPSLA 2014 Call for Artifacts. `http://2014.splashcon.org/track/splash2014-artifacts`.

[3] Oracle. Java SE Downloads. `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

[4] Oracle. Launching a Java application. `http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/java.html`.

[5] Oracle. PATH and CLASSPATH. `http://docs.oracle.com/javase/tutorial/essential/environment/paths.html`.