Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity (Artifact)

This document introduces how to setup our artifact and use it to reproduce the results of the accompanying paper.

Part I: Getting Started Guide

1 Basic Requirements

- A 64-bit Ubuntu system (version 14.04 – 18.04)
- A Python 2 interpreter (version 2.7+), if absent, install it by

```
$ sudo apt install python
```

- A Java 1.8 distribution, if absent, install it by

```
$ sudo apt install openjdk-8-jdk openjdk-8-jre
```

2 Installing the PA-Datalog Engine

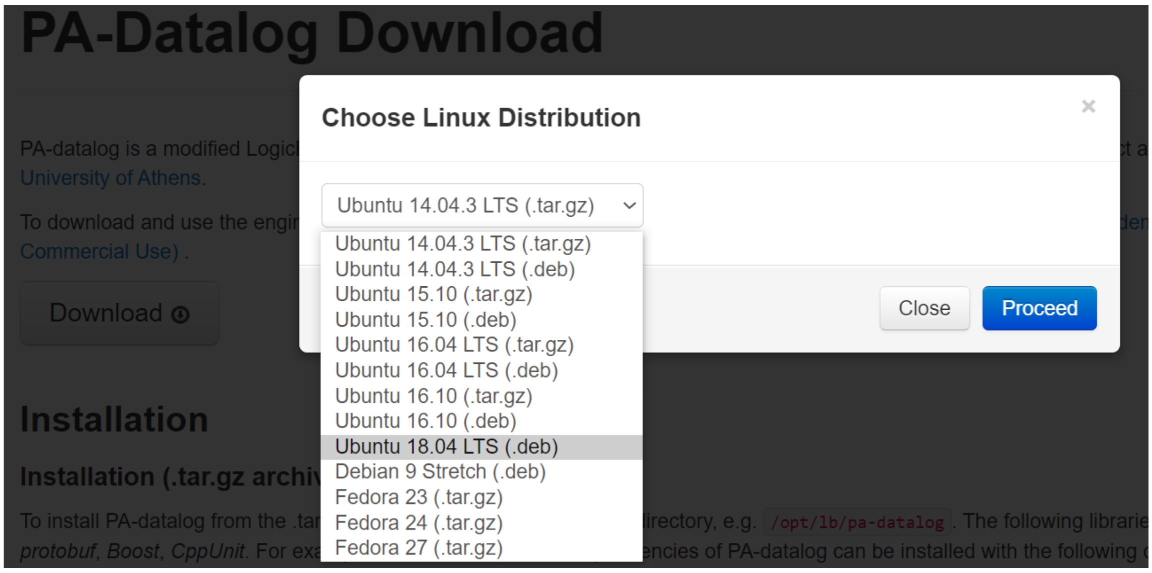
To run Doop framework, you need to install a PA-Datalog engine for interpreting the Datalog rules used in Doop. Since we cannot re-distribute our copy of this engine to others due to the *licensing issues*, you need to install the engine by yourself. Below we introduce how to install and setup the engine.

Installation Firstly, please download the engine from this page:

http://snf-705535.vm.okeanos.grnet.gr/agreement.html

Note that to download the engine, you must accept the academic license (which will automatically pop up after you click the "Download" and "Proceed" buttons).

For each Ubuntu system, there are one or two files, i.e., a `.deb` file and/or a `.tar.gz` file, corresponding to two installation approaches (as shown below). For convenience, we recommend `.deb` package installation. For example, if you use Ubuntu 18.04, then just select "Ubuntu 18.04 LTS (.deb)" and proceed.



PA-Datalog Download

PA-datalog is a modified Logic

University of Athens.

To download and use the engi

Commercial Use) .

Installation

Installation (.tar.gz archi

To install PA-datalog from the .ta … irectory, e.g. `/opt/lb/pa-datalog`. The following librarie

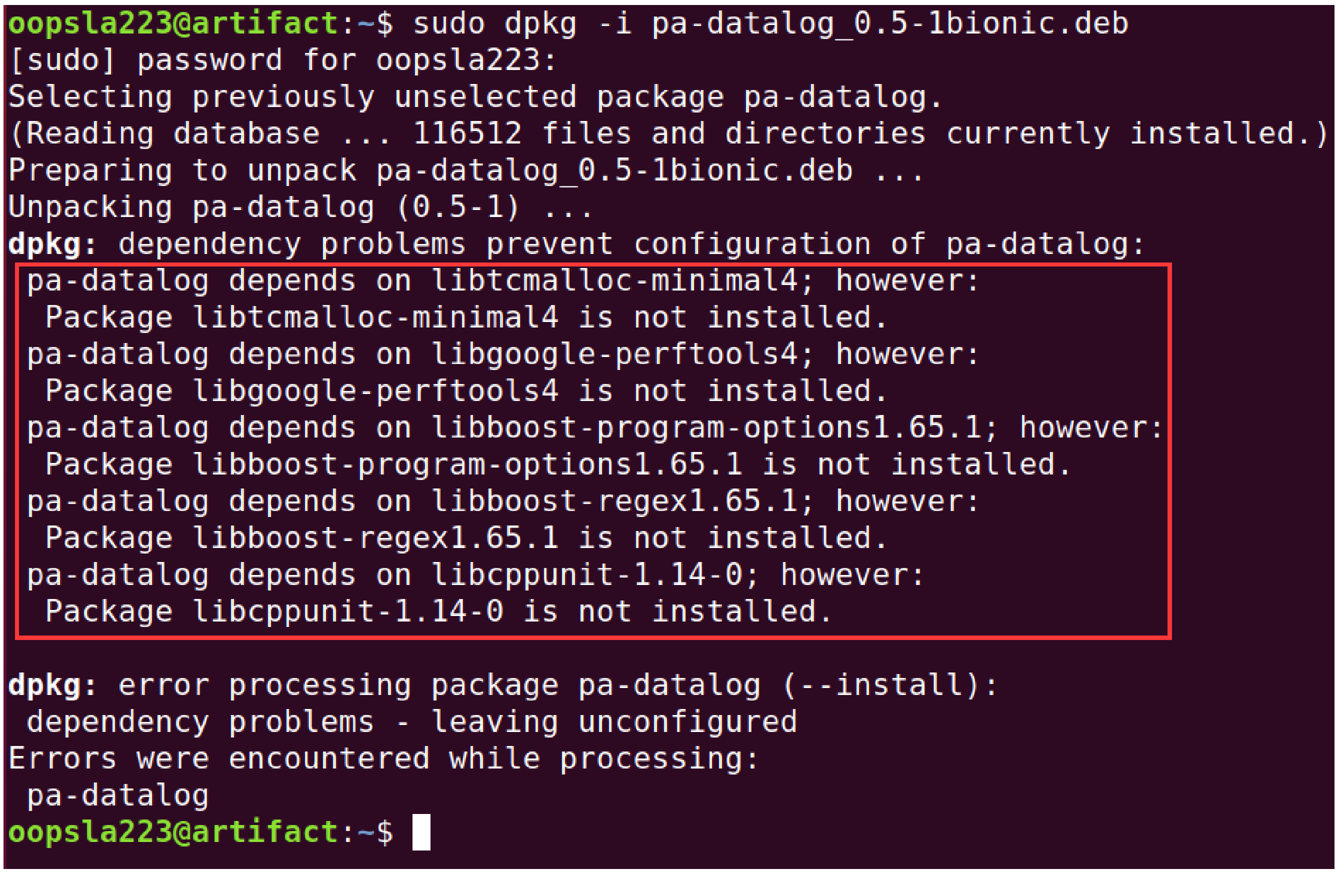
protobuf, *Boost*, *CppUnit*. For exa … encies of PA-datalog can be installed with the following

Choose Linux Distribution

To install the engine through `.deb` file (e.g., `pa-datalog_0.5-1bionic.deb` for Ubuntu 18.04), just run the following two commands:

```
$ sudo dpkg -i pa-datalog_0.5-1bionic.deb
$ sudo apt-get install -f
```

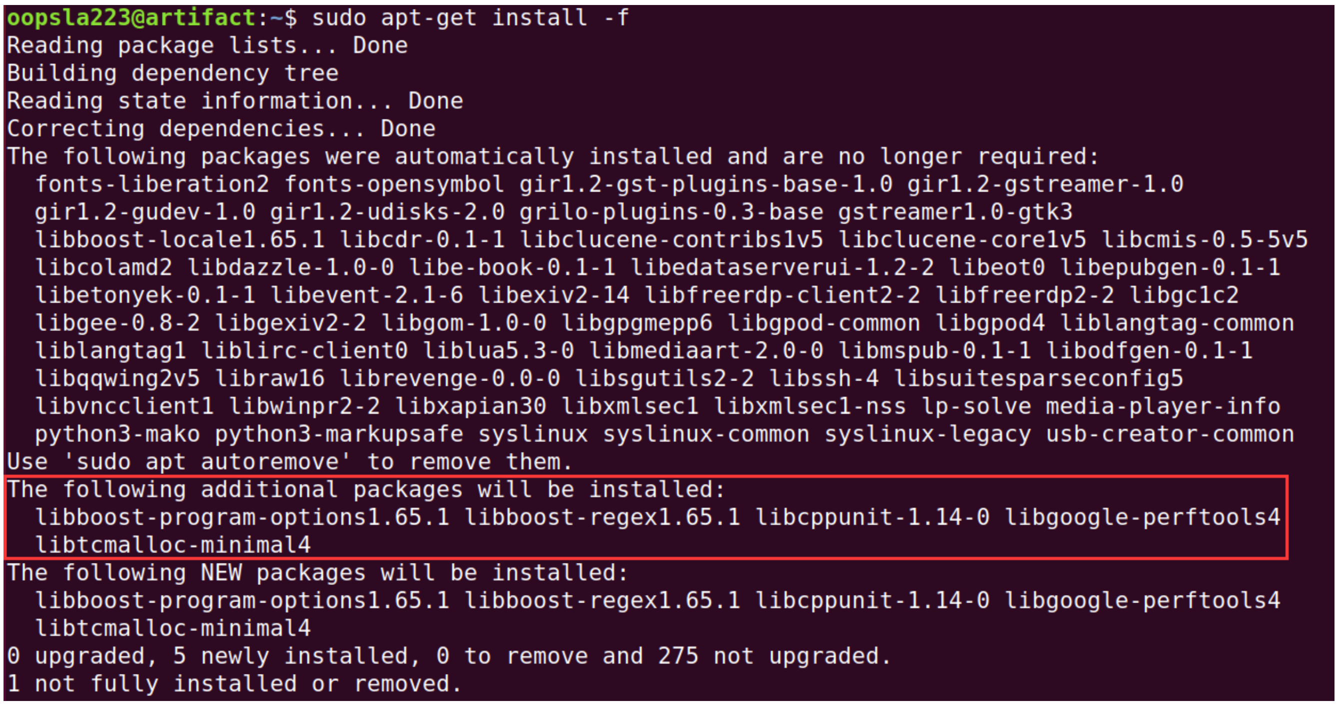
Note that the first command may report some missing packages as shown below:



```
oopsla223@artifact:~$ sudo dpkg -i pa-datalog_0.5-1bionic.deb
[sudo] password for oopsla223:
Selecting previously unselected package pa-datalog.
(Reading database ... 116512 files and directories currently installed.)
Preparing to unpack pa-datalog_0.5-1bionic.deb ...
Unpacking pa-datalog (0.5-1) ...
dpkg: dependency problems prevent configuration of pa-datalog:
 pa-datalog depends on libtcmalloc-minimal4; however:
  Package libtcmalloc-minimal4 is not installed.
 pa-datalog depends on libgoogle-perftools4; however:
  Package libgoogle-perftools4 is not installed.
 pa-datalog depends on libboost-program-options1.65.1; however:
  Package libboost-program-options1.65.1 is not installed.
 pa-datalog depends on libboost-regex1.65.1; however:
  Package libboost-regex1.65.1 is not installed.
 pa-datalog depends on libcppunit-1.14-0; however:
  Package libcppunit-1.14-0 is not installed.

dpkg: error processing package pa-datalog (--install):
 dependency problems - leaving unconfigured
Errors were encountered while processing:
 pa-datalog
oopsla223@artifact:~$
```

This is expected, and these packages will be automatically installed by the second command:



```
oopsla223@artifact:~$ sudo apt-get install -f
Reading package lists... Done
Building dependency tree
Reading state information... Done
Correcting dependencies... Done
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0
  gir1.2-gudev-1.0 gir1.2-udisks-2.0 grilo-plugins-0.3-base gstreamer1.0-gtk3
  libboost-locale1.65.1 libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5
  libcolamd2 libdazzle-1.0-0 libe-book-0.1-1 libedataserverui-1.2-2 libeot0 libepubgen-0.1-1
  libetonyek-0.1-1 libevent-2.1-6 libexiv2-14 libfreerdp-client2-2 libfreerdp2-2 libgc1c2
  libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6 libgpod-common libgpod4 liblangtag-common
  liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1 libodfgen-0.1-1
  libqqwing2v5 libraw16 librevenge-0.0-0 libsgutils2-2 libssh-4 libsuitesparseconfig5
  libvncclient1 libwinpr2-2 libxapian30 libxmlsec1 libxmlsec1-nss lp-solve media-player-info
  python3-mako python3-markupsafe syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libboost-program-options1.65.1 libboost-regex1.65.1 libcppunit-1.14-0 libgoogle-perftools4
  libtcmalloc-minimal4
The following NEW packages will be installed:
  libboost-program-options1.65.1 libboost-regex1.65.1 libcppunit-1.14-0 libgoogle-perftools4
  libtcmalloc-minimal4
0 upgraded, 5 newly installed, 0 to remove and 275 not upgraded.
1 not fully installed or removed.
```

After executing the two commands above, the PA-Datalog engine now should be in directory `/opt/lb/pa-datalog` .

Setup To make the binaries of PA-Datalog available for use in the shell, you need to:

1. Set `JAVA_HOME` to an appropriate location for Java:

   ```
   $ export JAVA_HOME=/path/to/java-8
   ```

   Please set a correct path for `JAVA_HOME`. To obtain the correct path to Java home, you could use command:

   ```
   $ java -XshowSettings:properties 2>&1 | grep 'java.home'
   ```

2. Source the `lb-env-bin.sh` script in your shell to set required paths:

   ```
   $ source /opt/lb/pa-datalog/lb-env-bin.sh
   ```

Then the PA-Datalog engine is ready to use. You may include the above commands in your startup script such as `.bashrc`, so that you don't need to setup the engine again when you open a new shell.

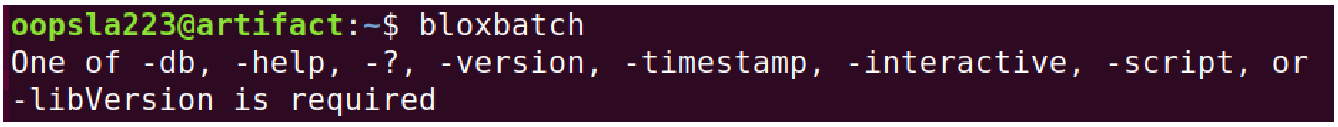
Now you can start evaluating our artifact.

3 Basic Testing

Testing PA-Datalog Installation A simple way to test the installation of PA-Datalog engine is just to run this command:

```
$ bloxbatch
```

If the outputs of the command are the same as below and no errors occur, then it is expected that the PA-Datalog engine has been successfully installed.



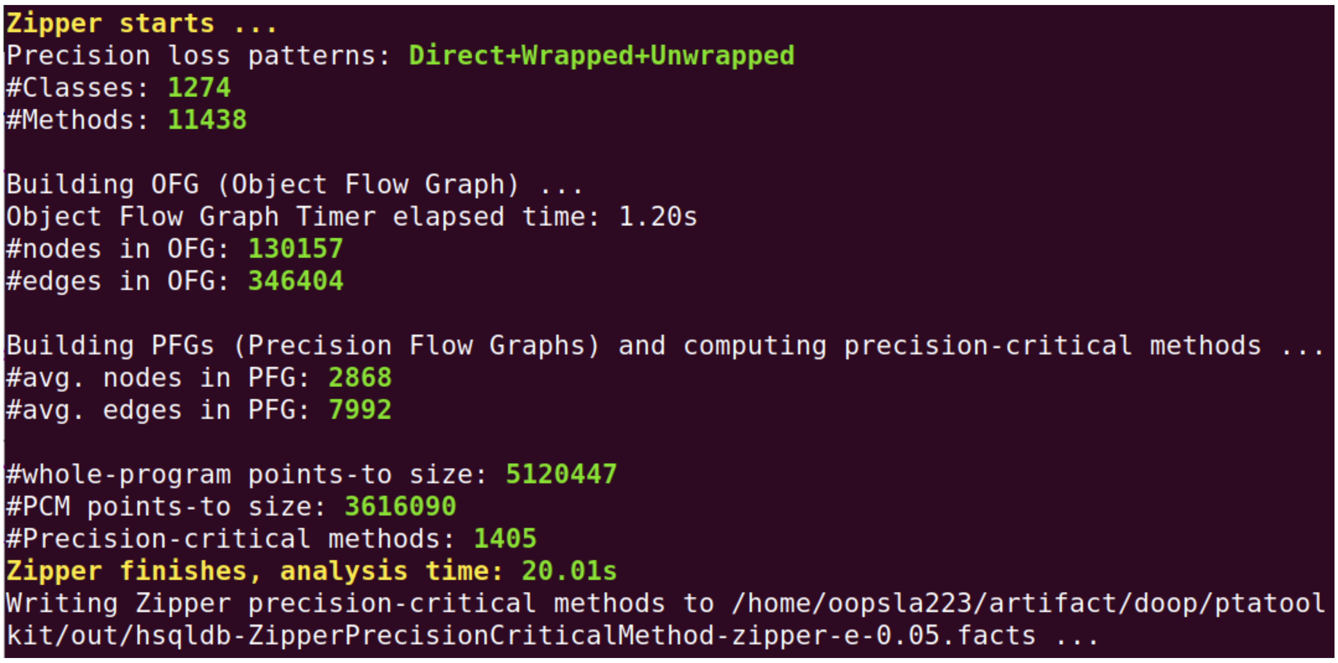
```
oopsla223@artifact:~$ bloxbatch
One of -db, -help, -?, -version, -timestamp, -interactive, -script, or
-libVersion is required
```

Testing the Artifact To test whether the artifact has been successfully setup, please first change your current directory to where you unpacked the artifact package. Then you can use the following command to analyze `hsqldb` by Zipper-e-guided pointer analysis:

```
$ ./run.py zipper-e hsqldb
```

We use `zipper-e` (one of the selective approaches used in our paper) for testing as it is fast and it can cover the installations of Python, Java, PA-Datalog engine as well as the artifact itself.

This command will invoke context-insensitive pointer analysis (as pre-analysis of Zipper-e), Zipper-e, and Zipper-e-guided context-sensitive pointer analysis in order, and it would take about 5 minutes to finish. If the outputs of the command contain the Zipper-e outputs like this:



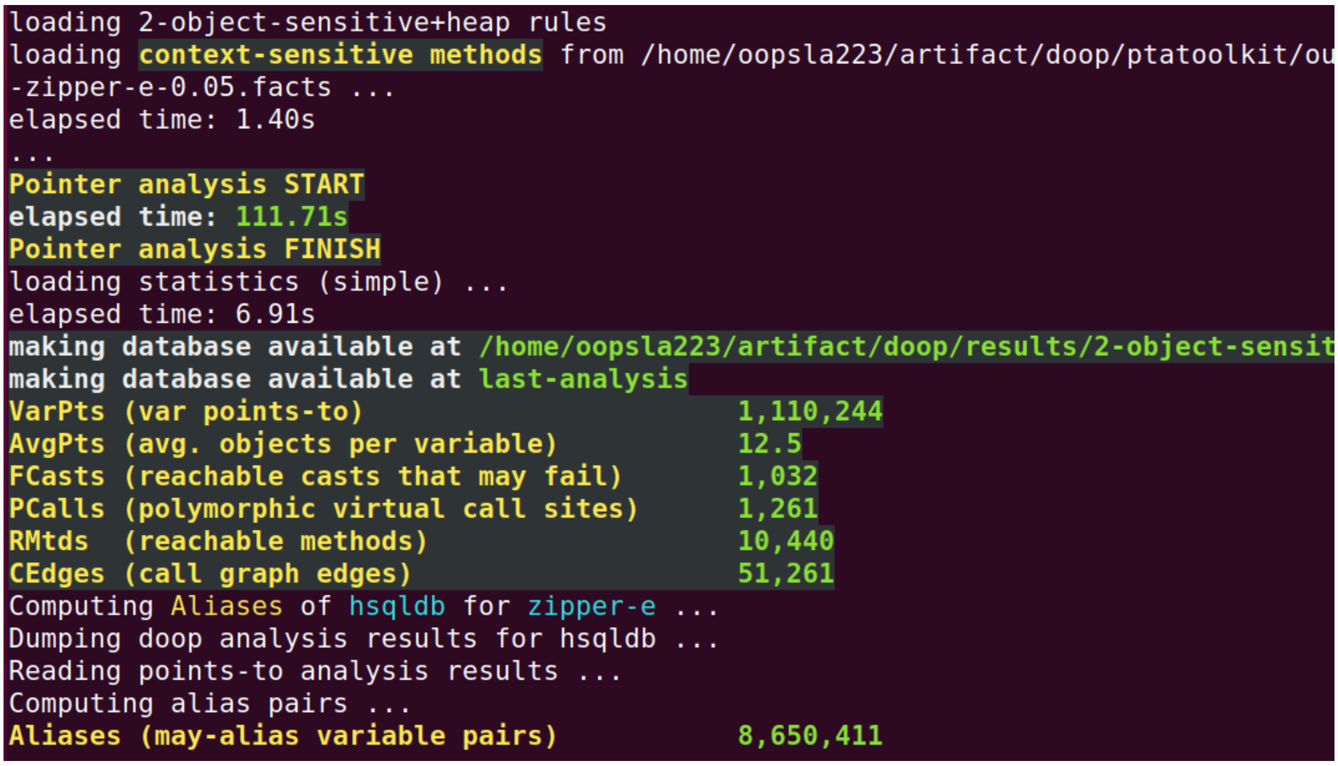
```
Zipper starts ...
Precision loss patterns: Direct+Wrapped+Unwrapped
#Classes: 1274
#Methods: 11438

Building OFG (Object Flow Graph) ...
Object Flow Graph Timer elapsed time: 1.20s
#nodes in OFG: 130157
#edges in OFG: 346404

Building PFGs (Precision Flow Graphs) and computing precision-critical methods ...
#avg. nodes in PFG: 2868
#avg. edges in PFG: 7992

#whole-program points-to size: 5120447
#PCM points-to size: 3616090
#Precision-critical methods: 1405
Zipper finishes, analysis time: 20.01s
Writing Zipper precision-critical methods to /home/oopsla223/artifact/doop/ptatool
kit/out/hsqldb-ZipperPrecisionCriticalMethod-zipper-e-0.05.facts ...
```

and finally the context-sensitive pointer analysis outputs like this:



```
loading 2-object-sensitive+heap rules
loading context-sensitive methods from /home/oopsla223/artifact/doop/ptatoolkit/ou
-zipper-e-0.05.facts ...
elapsed time: 1.40s
...
Pointer analysis START
elapsed time: 111.71s
Pointer analysis FINISH
loading statistics (simple) ...
elapsed time: 6.91s
making database available at /home/oopsla223/artifact/doop/results/2-object-sensit
making database available at last-analysis
VarPts (var points-to)                      1,110,244
AvgPts (avg. objects per variable)          12.5
FCasts (reachable casts that may fail)      1,032
PCalls (polymorphic virtual call sites)     1,261
RMtds  (reachable methods)                  10,440
CEdges (call graph edges)                   51,261
Computing Aliases of hsqldb for zipper-e ...
Dumping doop analysis results for hsqldb ...
Reading points-to analysis results ...
Computing alias pairs ...
Aliases (may-alias variable pairs)          8,650,411
```

then the artifact should have been successfully setup, and the user should have no technical difficulties with the rest of the artifact.

Part II: Step-by-Step Instructions

1 Introduction

This artifact is provided to reproduce the results of research questions (RQ1 and RQ2) in Section 6 of our companion paper, i.e., to reproduce the results in:

- Table 1
- Figure 7
- Table 2
- Figure 8

Note that in our paper, our experiments are carried out on a machine with 128GB of RAM. Using a machine with a smaller RAM may cause some analyses to run more slowly or even to the point of being unscalable. As a result, the results concerning analysis times need to be checked with caution due to the differences in the running environments used.

All the time-irrelevant data in the above tables and figures can be reproduced by this artifact. As the companion paper contains many experiments for different kinds of pointer analyses and configurations of our tool, **to ease the users to examine all the experimental results or any results they are particularly interested in, we specially design and provide different commands to make such examinations convenient**.

This document gives comprehensive step-by-step instructions on how to use this artifact to conveniently reproduce the results in the companion paper. For setting up the artifact, please refer to *Getting Started Guide*.

2 Content of Artifact

This artifact includes:

- `README.pdf` : the document of this artifact
- `LICENSE.txt` : the license of this artifact
- `run.py` : a Python script for driving all the provided analyses
- `doop` : the folder containing Doop framework (a state-of-the-art pointer analysis framework for Java). It also contains the source code and implementation of Baton (our tool, as well as Collection, Zipper-e and Scaler which are used by Baton), the Java programs and the library to be analyzed in the evaluation.

3 Running Experiments

To run the experiments, please first follow the instructions in *Getting Started Guide* to setup, and then change your current directory to where you unpacked the artifact package.

3.1 Running Analyses in Table 1

Running pointer analyses produces the results in **Table 1** and **Figure 7** (which is a visualized presentation of the fifth column of Table 1). The command to run pointer analysis is:

```
$ ./run.py <ANALYSIS> <PROGRAM>
```

The `<ANALYSIS>` can be one of the following pointer analyses evaluated in Table 1:

`ci` , `collection` , `zipper-e` , `scaler` , `baton-unity`

The `<PROGRAM>` can be one of the following Java programs analyzed in Table 1:

`hsqldb` , `galleon` , `jedit` , `soot` , `gruntspud` , `heritrix` , `pmd` , `jython` , `jasperreports` , `eclipse` , `briss` , `columba` .

For example, to use `ci` to analyze `hsqldb` , type:

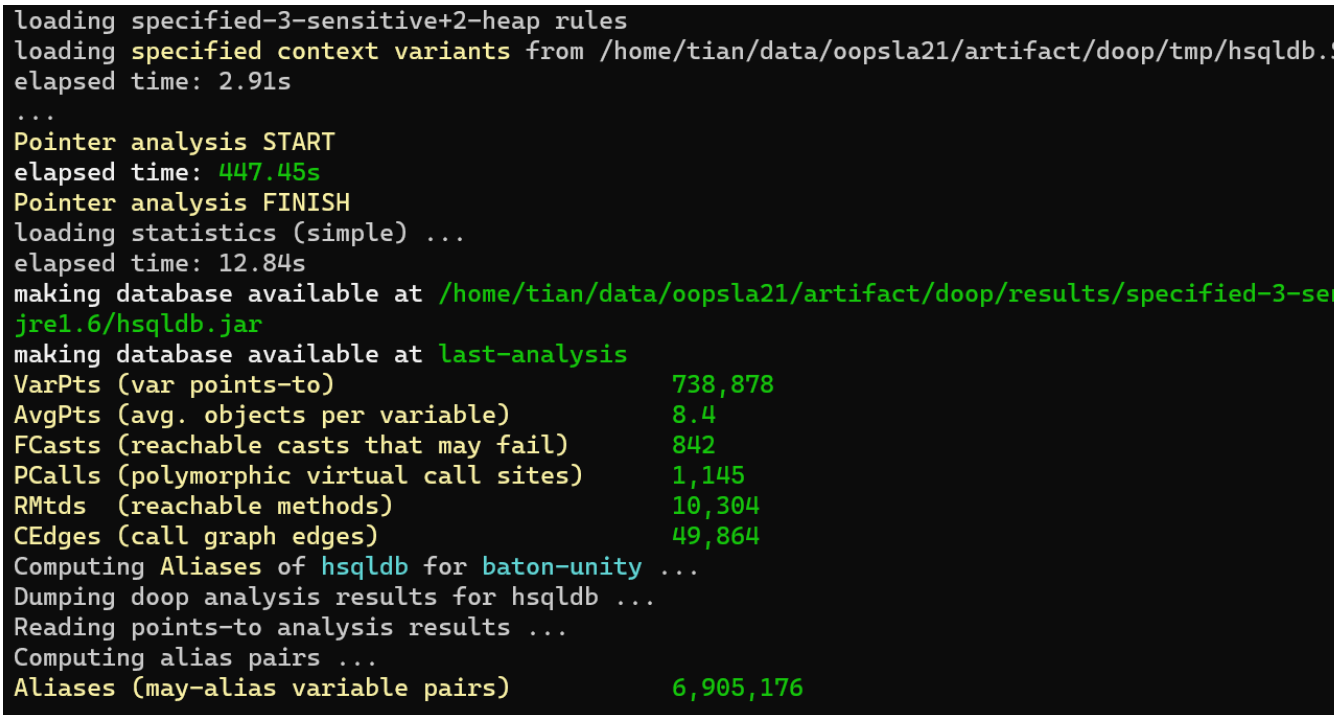
```
$ ./run.py ci hsqldb
```

To use `baton-unity` (i.e., our tool) to analyze `hsqldb` , type:

```
$ ./run.py baton-unity hsqldb
```

`baton-unity` will first obtain the results from the input selective approaches (`collection` , `zipper-e` and `scaler` in our setting), and then picks the "most precise configuration" and uses it to guide the context-sensitive pointer analysis.

When running pointer analysis, the data in Table 1 and Figure 7, i.e., the pointer analysis time and precision metrics used (`VarPts` , `AvgPts` , `FCasts` , `PCalls` , `RMtds` , `CEdges` , and `Aliases`) will be printed on the screen as shown below:



```
loading specified-3-sensitive+2-heap rules
loading specified context variants from /home/tian/data/oopsla21/artifact/doop/tmp/hsqldb.
elapsed time: 2.91s
...
Pointer analysis START
elapsed time: 447.45s
Pointer analysis FINISH
loading statistics (simple) ...
elapsed time: 12.84s
making database available at /home/tian/data/oopsla21/artifact/doop/results/specified-3-se
jre1.6/hsqldb.jar
making database available at last-analysis
VarPts (var points-to)                       738,878
AvgPts (avg. objects per variable)           8.4
FCasts (reachable casts that may fail)       842
PCalls (polymorphic virtual call sites)      1,145
RMtds  (reachable methods)                   10,304
CEdges (call graph edges)                    49,864
Computing Aliases of hsqldb for baton-unity ...
Dumping doop analysis results for hsqldb ...
Reading points-to analysis results ...
Computing alias pairs ...
Aliases (may-alias variable pairs)           6,905,176
```

For convenience, we provide the option to run **all five pointer analyses** in Table 1 for a certain program. For example, to run all pointer analyses for `hsqldb` , type:

```
$ ./run.py -table1 hsqldb
```

3.2 Running Analyses in Table 2

Running pointer analyses produces the results in **Table 2** and **Figure 8** (which is a visualized presentation of the fifth column of Table 2). The command to run pointer analysis is as follows (`[]` means *optional*):

```
$ ./run.py <ANALYSIS> [-table2] <PROGRAM>
```

The `<ANALYSIS>` can be one of the following pointer analyses evaluated in Table 2:

`ci`, `zipper-e`, `scaler`, `baton-relay`.

The `<PROGRAM>` can be one of the following Java programs analyzed in Table 2:

`eclipse`, `briss`, `pmd`, `jedit`, `h2`, `gruntspud`.

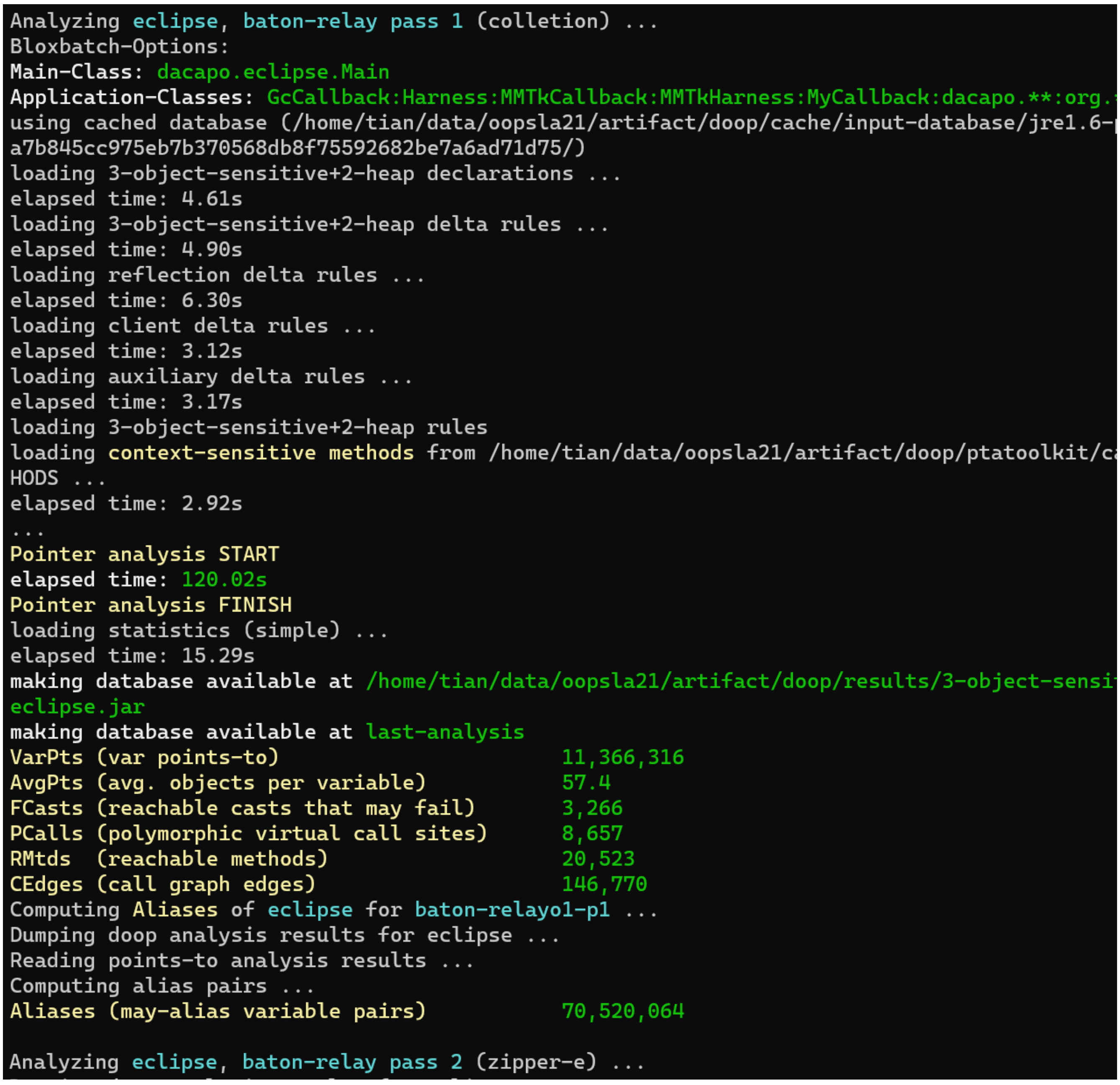
Note that in Table 2, the configurations of `zipper-e` and `scaler` are different from the ones in Table 1 as mentioned in our companion paper. Thus, when running these two analyses (i.e., `zipper-e` and `scaler`) to produce the data in Table 2, you need to append option `-table2` after `<ANALYSIS>`. For example, to analyze `eclipse` by `scaler` in Table 2, type:

```
$ ./run.py scaler -table2 eclipse
```

To analyze `eclipse` by `baton-relay` (i.e., our tool), type:

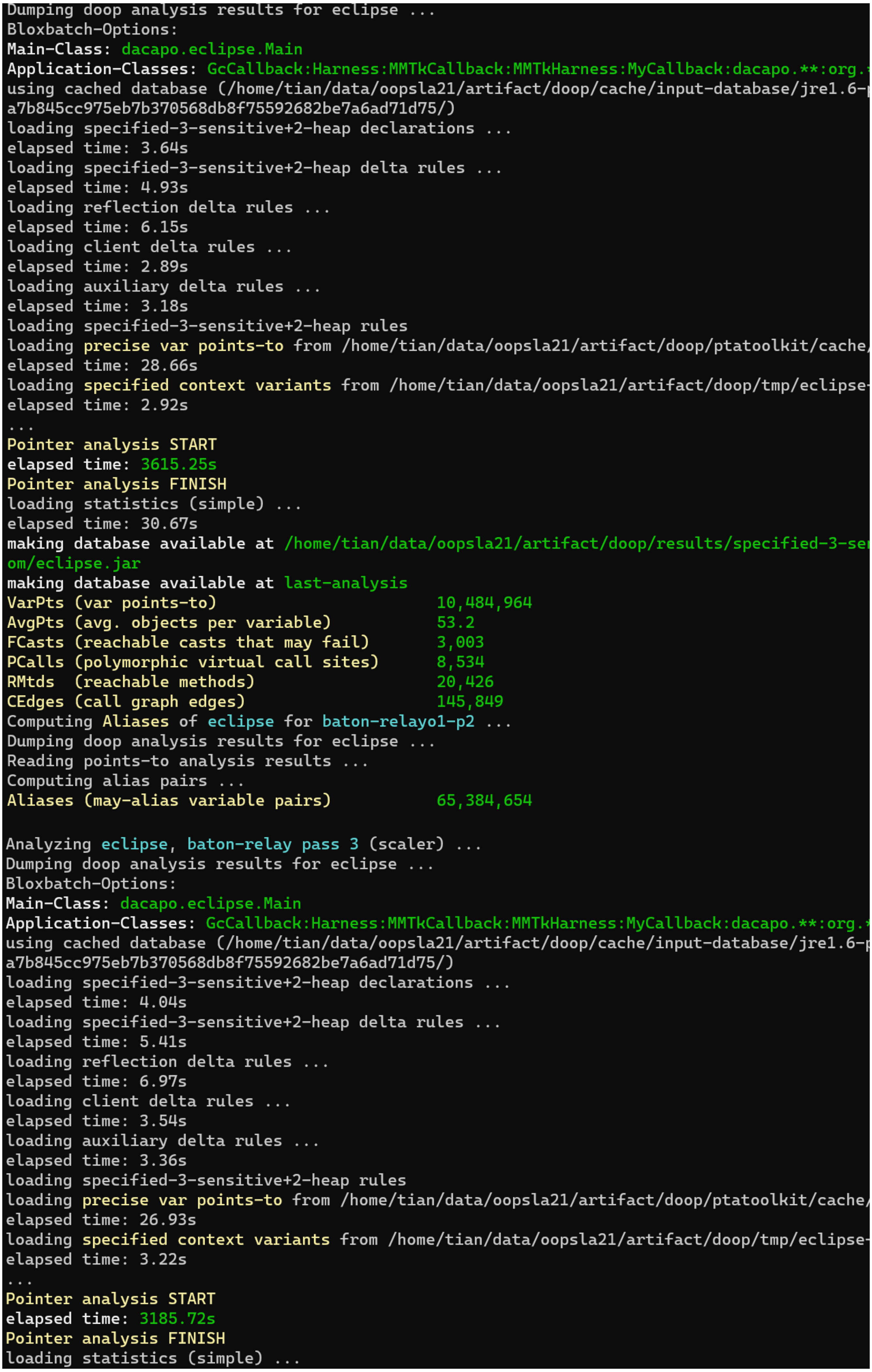
```
$ ./run.py baton-relay eclipse
```

`baton-relay` will first obtain the results from the input selective approaches, selects the proper configuration for each pass, and then runs the three passes (one context-sensitive pointer analysis for each pass) as shown below:



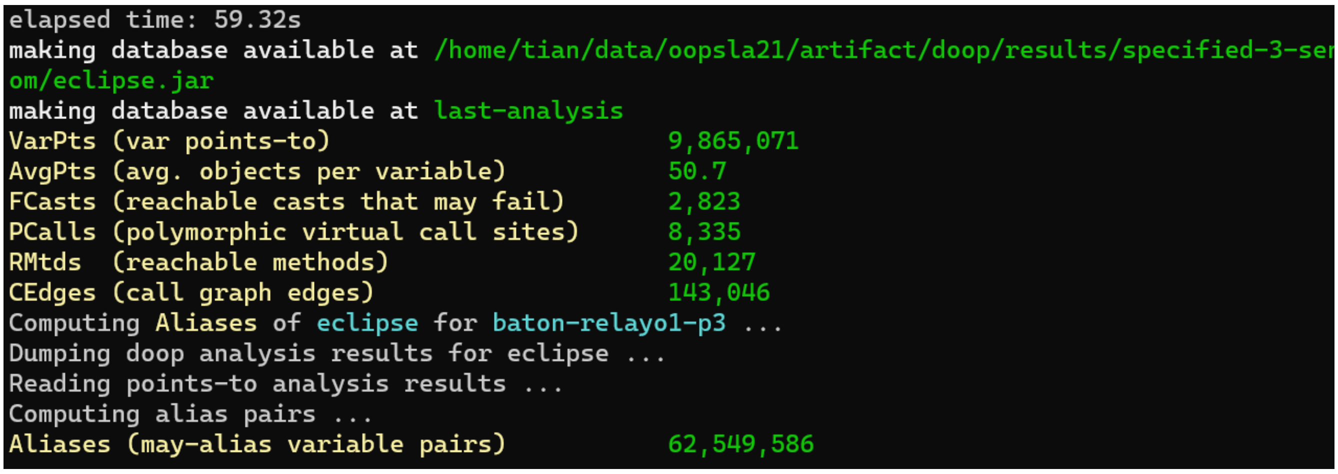
```
Analyzing eclipse, baton-relay pass 1 (colletion) ...
Bloxbatch-Options:
Main-Class: dacapo.eclipse.Main
Application-Classes: GcCallback:Harness:MMTkCallback:MMTkHarness:MyCallback:dacapo.**:org.
using cached database (/home/tian/data/oopsla21/artifact/doop/cache/input-database/jre1.6-
a7b845cc975eb7b370568db8f75592682be7a6ad71d75/)
loading 3-object-sensitive+2-heap declarations ...
elapsed time: 4.61s
loading 3-object-sensitive+2-heap delta rules ...
elapsed time: 4.90s
loading reflection delta rules ...
elapsed time: 6.30s
loading client delta rules ...
elapsed time: 3.12s
loading auxiliary delta rules ...
elapsed time: 3.17s
loading 3-object-sensitive+2-heap rules
loading context-sensitive methods from /home/tian/data/oopsla21/artifact/doop/ptatoolkit/c
HODS ...
elapsed time: 2.92s
...
Pointer analysis START
elapsed time: 120.02s
Pointer analysis FINISH
loading statistics (simple) ...
elapsed time: 15.29s
making database available at /home/tian/data/oopsla21/artifact/doop/results/3-object-sensi
eclipse.jar
making database available at last-analysis
VarPts (var points-to)                      11,366,316
AvgPts (avg. objects per variable)          57.4
FCasts (reachable casts that may fail)      3,266
PCalls (polymorphic virtual call sites)     8,657
RMtds  (reachable methods)                  20,523
CEdges (call graph edges)                   146,770
Computing Aliases of eclipse for baton-relayo1-p1 ...
Dumping doop analysis results for eclipse ...
Reading points-to analysis results ...
Computing alias pairs ...
Aliases (may-alias variable pairs)          70,520,064

Analyzing eclipse, baton-relay pass 2 (zipper-e) ...
```



```
Dumping doop analysis results for eclipse ...
Bloxbatch-Options:
Main-Class: dacapo.eclipse.Main
Application-Classes: GcCallback:Harness:MMTkCallback:MMTkHarness:MyCallback:dacapo.**:org.
using cached database (/home/tian/data/oopsla21/artifact/doop/cache/input-database/jre1.6-
a7b845cc975eb7b370568db8f75592682be7a6ad71d75/)
loading specified-3-sensitive+2-heap declarations ...
elapsed time: 3.64s
loading specified-3-sensitive+2-heap delta rules ...
elapsed time: 4.93s
loading reflection delta rules ...
elapsed time: 6.15s
loading client delta rules ...
elapsed time: 2.89s
loading auxiliary delta rules ...
elapsed time: 3.18s
loading specified-3-sensitive+2-heap rules
loading precise var points-to from /home/tian/data/oopsla21/artifact/doop/ptatoolkit/cache/
elapsed time: 28.66s
loading specified context variants from /home/tian/data/oopsla21/artifact/doop/tmp/eclipse-
elapsed time: 2.92s
...
Pointer analysis START
elapsed time: 3615.25s
Pointer analysis FINISH
loading statistics (simple) ...
elapsed time: 30.67s
making database available at /home/tian/data/oopsla21/artifact/doop/results/specified-3-se
om/eclipse.jar
making database available at last-analysis
VarPts (var points-to)                    10,484,964
AvgPts (avg. objects per variable)        53.2
FCasts (reachable casts that may fail)    3,003
PCalls (polymorphic virtual call sites)   8,534
RMtds  (reachable methods)                20,426
CEdges (call graph edges)                 145,849
Computing Aliases of eclipse for baton-relayo1-p2 ...
Dumping doop analysis results for eclipse ...
Reading points-to analysis results ...
Computing alias pairs ...
Aliases (may-alias variable pairs)        65,384,654

Analyzing eclipse, baton-relay pass 3 (scaler) ...
Dumping doop analysis results for eclipse ...
Bloxbatch-Options:
Main-Class: dacapo.eclipse.Main
Application-Classes: GcCallback:Harness:MMTkCallback:MMTkHarness:MyCallback:dacapo.**:org.
using cached database (/home/tian/data/oopsla21/artifact/doop/cache/input-database/jre1.6-
a7b845cc975eb7b370568db8f75592682be7a6ad71d75/)
loading specified-3-sensitive+2-heap declarations ...
elapsed time: 4.04s
loading specified-3-sensitive+2-heap delta rules ...
elapsed time: 5.41s
loading reflection delta rules ...
elapsed time: 6.97s
loading client delta rules ...
elapsed time: 3.54s
loading auxiliary delta rules ...
elapsed time: 3.36s
loading specified-3-sensitive+2-heap rules
loading precise var points-to from /home/tian/data/oopsla21/artifact/doop/ptatoolkit/cache/
elapsed time: 26.93s
loading specified context variants from /home/tian/data/oopsla21/artifact/doop/tmp/eclipse-
elapsed time: 3.22s
...
Pointer analysis START
elapsed time: 3185.72s
Pointer analysis FINISH
loading statistics (simple) ...
```



```
elapsed time: 59.32s
making database available at /home/tian/data/oopsla21/artifact/doop/results/specified-3-ser
om/eclipse.jar
making database available at last-analysis
VarPts (var points-to)                      9,865,071
AvgPts (avg. objects per variable)          50.7
FCasts (reachable casts that may fail)      2,823
PCalls (polymorphic virtual call sites)     8,335
RMtds  (reachable methods)                  20,127
CEdges (call graph edges)                   143,046
Computing Aliases of eclipse for baton-relayo1-p3 ...
Dumping doop analysis results for eclipse ...
Reading points-to analysis results ...
Computing alias pairs ...
Aliases (may-alias variable pairs)          62,549,586
```

Similar to Table 1, we also provide a convenient option to run all pointer analyses in Table 2 for a certain program. For example, to run all pointer analyses for `eclipse`, type:

```
$ ./run.py -table2 eclipse
```

3.3 Running All Experiments

We provide an option to run Baton and all other pointer analyses for all programs, to produce all data in Table 1 (including Figure 7), Table 2 (including Figure 8) in one shot:

```
$ ./run.py -all
```

Although it is convenient, we recommend the user to try the commands in Sections 3.1 and 3.2 first, since this option is both time- and space-consuming by running a large amount of sets of experiments. In our experimental settings, this option spends about 50 hours and consumes about 290 GB of disk space as Doop keeps flushing all its results to the disk.