OOPSLA'25 Round2 Artifact Evaluation Document

1. Introduction

This artifact supports our OOPSLA 2025 paper titled **[Towards a Theoretically-backed and Practical Framework for Selective Object-Sensitive Pointer Analysis]**. It provides the implementation and experimental scripts needed to reproduce the core evaluation results presented in the paper. The artifact contains:

- Our implementation of the proposed approach, **Moon**.
- Implementation of other approaches: **Cut-ShortCut, Zipper, Conch, DebloaterX**
- Scripts to run experiments on benchmark configurations on **Context-Insensitivity, 2Object-Sensitive and 3Object-Sensitive**.
- Output logs and raw data used to generate key tables and figures.

> Note: Our paper has received a *Minor Revision* decision. The required changes have minimal impact on the artifact. Therefore, unless specific modification requests arise during the Artifact Evaluation process, the current artifact can be considered as nearly final.

1.1 Supported Claims

Below we list key claims from the paper and describe how they are supported by the artifact:

- **Claim 1**: Our approach achieves a better trade-off between precision and efficiency for pointer analysis than baselines (Table 1, Table 2).
- **Claim 2**: Our approach identifies less context-sensitive heaps compared to DebloaterX (Figure 8).
- **Claim 3**: Heuristic time comparison shows efficiency benefits (Table 3).
- **Claim 4**: The details of Precision-relevant heaps identified by Moon (Figure 11).

All the claims can be derived from the output of our artifact, details in Section 4.

1.2 Omitted Claims

Some summary figures: Figures 9, 10 and 12 aggregate across many full experimental runs, which requires a large amount of time and resources, so we do not reproduce the full calculation pipeline in the artifact.

2. Hardware Dependencies

All experiments were originally conducted on a desktop machine equipped with:

- **CPU**: Intel(R) Core(TM) i9-14900K
- **RAM**: 192 GB
- **OS**: Ubuntu 22.04 LTS

A **high-memory system (≥192 GB RAM)** is recommended for executing the full benchmark suite, especially for large projects. Running on machines with **lower memory (e.g., 32–64 GB RAM)** may result in out-of-memory errors or cause certain large benchmarks to become unscalable.

3. Get Started

3.1 Software Requirements

- **JDK 17**
- **Python 3.9** with `pandas` installed

> **Note:** The `python` command may not be available by default as Python 3.x typically installs the binary as `python3`. You may need to create an alias or symbolic link to redirect `python` to `python3`.

3.2 Basic Testing

To perform a basic functionality check, we recommend running our tool **Moon** on the benchmark **antlr** with **2-Object-Sensitivity**. This test ensures that the environment is correctly set up and the core analysis pipeline executes successfully.

Run the following command:

```
cd artifact
python run.py antlr 2o -print -cd -cda=MOON | grep "####"
```

This command invokes the Moon on the antlr benchmark and prints key metrics to the console. The `grep "####"` filter displays only the summary line of results.

Expected Output

You should see metrics like:

```
####Base: 89
####Recursive: 11
####HeuristicTime:4.22
####CSHeaps:100
####AllHeaps:9224
####T:                                                    2.99
####RM:                                                   7806
####CE:                                                   51264
####MFC:                                                  510
####PCS:                                                  1631
```

Note: The values of **HeuristicTime** and **T** may vary on your machine, as they represent execution times and are affected by hardware and system load.

4. Step by Step Instructions

This section describes how to reproduce the experimental results reported in our paper. It covers running all or selected benchmarks for different approaches, as well as ablation studies.

4.1 Running Benchmarks (for RQ1-RQ3)

4.1.1 Runing a specific benchmark for a specfic appraoch

This mode is recommended for quick testing and avoids the long runtimes of full benchmark suites. We suggest using small benchmarks such as **antlr**, **biojava**, **fop**, and **h2** for faster evaluation.

The key experimental metrics will be printed to the standard output, with each line prefixed by `"####"`.

Moon (Our tool)

```
cd artifact
python run.py {benchmark_name} {sensitivity} -print -cd -cda=MOON | grep "####"
```

- Available sensitivities: *2o* (2obj-sensitivity) and *3o* (3obj-sensitivity).
- Benchmark names are listed in *artifact/util/benchmark.py*.

Cut-Shortcut

```
cd artifact
python run.py {benchmark_name} csc -print | grep "####"
```

- Benchmark names are listed in *artifact/util/benchmark.py*.

Zipper

```
cd artifact
python run.py {benchmark_name} {sensitivity} -print | grep "####"
```

- Available sensitivities: *Z-2o* (2obj-sensitivity) and *Z-3o* (3obj-sensitivity).
- Benchmark names are listed in *artifact/util/benchmark.py*.

Conch

```
cd artifact
python run.py {benchmark_name} {sensitivity} -print -cd -cda=CONCH | grep "####"
```

- Available sensitivities: *2o* and *3o*.
- Benchmark names are listed in *artifact/util/benchmark.py*.

DebloaterX

```
cd artifact
python run.py {benchmark_name} {sensitivity} -print -cd -
cda=DEBLOATERX | grep "####"
```

- Available sensitivities: *2o* and *3o*.
- Benchmark names are listed in *artifact/util/benchmark.py*.

Output Details

- The output contains metrics within a subset of the following ones:
  - **T**: Time of selective pointer analysis (seconds)
  - **RM**: Number of Reachable Methods
  - **CE**: Number of Call Edges
  - **MFC**: Number of May Fail Cast
  - **PCS**: Number of Polynomial CallSite
  - **HeuristicTime**: Time spent on heuristic computation
  - **CSHeaps**: Number of context-sensitive heaps
  - **AllHeaps**: Total number of heaps
  - **CSHeapRatio**: Proportion of CS heaps
  - **Base**, **Recursive**: Number of CS heaps classified as base or recursive
  - **BaseRatio**, **RecurRatio**: Ratios of base/recursive heaps to total CS heaps

4.1.2 Running all benchmarks for all approaches

Warning: this is gonna take a very long time (>5h)

```
cd artifact
python qilinDriverForAll.py
```

This command runs for all 30 benchmarks in our paper for approaches:

- Context-insensitive pointer analysis.
- Cut-Shortcut.

- Zipper (2obj-sensitive and 3obj-sensitive).
- Conch (2obj-sensitive and 3obj-sensitive).
- DebloaterX (2obj-sensitive and 3obj-sensitive).
- Moon (2obj-sensitive and 3obj-sensitive).

Output Details

All the key metrics are saved as CSV files in the directory: `artifact/result/`.

Each CSV file is named according to the following conventions:

- For **Zipper**, **Conch**, **DebloaterX**, and **Moon**: {benchmark}_{approach}_{sensitivity}.csv
- For **Context-Insensitive** baseline: {benchmark}_ci.csv
- For **Context-Sensitive** baseline: {benchmark}_PLAIN_{sensitivity}.csv
- For **Cut-Shortcut**: {benchmark}_csc.csv

4.2 Ablation Experiments (for RQ4)

4.2.1 Runing a specific benchmark for the varint Moonb.

```
cd artifact
python run.py {benchmark_name} {sensitivity} -print -cd -cda=MOONb | grep "####"
```

- Sensitivities: *2o* and *3o*.
- Benchmark names are listed in *artifact/util/benchmark.py*.

4.2.2 Running all benchmarks for the varint Moonb.

```
cd artifact
python qilinDriverForMoonAblation.py
```

Output: CSV files are saved in `artifact/result-abletion/`.

4.3 Output-to-Claim Mapping

4.3.1 Claim 1: (Table 1 and 2) a better trade-off between precision and efficiency

Check the output metrics (which is the same as the columns in Table 1 and 2):

- **T**: Total analysis time (seconds)
- **RM**: Number of Reachable Methods
- **CE**: Number of Call Edges
- **MFC**: Number of May Fail Cast
- **PCS**: Number of Polynomial Call-site

For approaches: Cut-ShortCut, Zipper, Conch, DebloaterX, Moon

and sensitivies: Context-insensitivity (ci), 2Obj-sensitivity (2o) and 3Obj-sensitivity (3o).

4.3.2 Claim 2: (Figure 8) our approach identifies less context-sensitive heaps compared to DebloaterX

Check the output metric:

- **CSHeapRatio**: Proportion of CS heaps

from approches: DebloaterX and Moon.

4.3.3 Claim 3: (Table 3) heuristic time comparison

Check the output metric:

- **HeuristicTime**: Time spent on heuristic computation

For approaches: Zipper, Conch, DebloaterX, Moon.

4.3.4 Claim 4: (Figure 11) the details of Precision-relevant heaps identified by Moon

Check the output metric:

- **Base**, **Recursive**: Number of CS heaps classified as base or recursive
- **BaseRatio**, **RecurRatio**: Ratios of base/recursive heaps to total CS heaps

For approach: Moon.

5. Reusability Guide

This section describes the core reusable components of the artifact, how to adapt the artifact to new inputs or use cases, and the limitations of reuse.

5.1 Core Reusable Components

The main reusable component is our implementation of **Moon**, located in: **qilin/pta/toolkit/moon/**.

The primary entry point is: **qilin/pta/toolkit/moon/Moon.java**.

The implementation of our tool Moon is integrated as a module into the **Qilin** pointer analysis framework and can be reused or extended as part of new analyses.

5.2 How to Adapt the Artifact

Since Moon is implemented within the **Qilin** pointer analysis framework, adapting to new inputs or use cases is straightforward and well-supported. For example:

- **New benchmarks**: New benchmarks can be easily analyzed using the option provided by Qilin framework.
- **Alternative analysis goals**: Moon can also be reused or extended for alternative static analysis goals, such as taint analysis or optimization since Moon benefits from Qilin's modular architecture and analysis pipeline, making it feasible to plug in custom logic or compose analyses.

For advanced integration or extension, please consult the official Qilin documentation.

5.3 Documentation and Entry Points

Since Moon is built atop the Qilin pointer analysis infrastructure, advanced users may refer to the official Qilin documentation, which includes detailed information about Qilin's architecture, intermediate representation (IR), and extensibility model.