Iterative-Epoch Online Cycle Elimination for Context-Free Language Reachability

PEI XU, University of Technology Sydney / UNSW Sydney, Australia
YUXIANG LEI, UNSW Sydney, Australia
YULEI SUI, UNSW Sydney, Australia
JINGLING XUE, UNSW Sydney, Australia

1 OVERVIEW

This artifact is for *Iterative-Epoch Online Cycle Elimination for Context-Free Language Reachability* by Pei Xu, Yuxiang Lei, Yulei Sui and Jingling Xue accepted to OOPSLA 2024.

Section 2 of this document provides instructions on setting up the environment and running a quick experiment to verify everything works. Section 3 provides instructions on reproducing our evaluation and varying the experiment through changes to source code, limits, and compiling new programs for analysis. Specifically, Section 3.3 shows how our artifact supports our claims.

1.1 Dependencies

This artifact requires Docker (tested with version 20.10.17 on an Ubuntu 20.04 machine) and the initial instructions for loading the image and starting a container assume a UNIX shell. An AMD64/x86-64 machine is also required. As our experiment performs analysis on large programs, a platform with an 128 GB's memory is preferred (16 GB is the minimum requirement to test and run small benchmarks using script `./shortTest.sh`).

1.2 Container Layout

Within the container (Section 2.1), our Iea is in the folder `$IEA_HOME`(i.e., `/root/IEA`). It has the following structure (some directories irrelevant to our purposes are omitted):

```
$IEA_HOME
|-- bench               # Benchmarks and scripts for our experiment
|   |-- graphtexts      # Transformed graphs
|   |   |-- aa          # graphs for aa
|   |   |-- vf          # graphs for vf
|   |-- run-aa.sh       # Script for performing the aa experiment
|   |-- run-vf.sh       # Script for performing the vf experiment
|   |-- shortTest-aa.sh # Script for performing the short aa experiment
|   |-- shortTest-vf.sh # Script for performing the short vf experiment
|   |-- fullTest.sh     # Script for performing the full experiment
|   |-- reTest.sh       # Script for performing failed cases
|   |-- tab1Gen.sh      # Script for generate table 1
|   |-- tab2Gen.sh      # Script for generate table 2
|   |-- tab3Gen.sh      # Script for generate table 3
|-- Release-build       # Build of IEA
`-- * (remainder)       # IEA sources.
```

2 GETTING STARTED GUIDE

In this section, we first load the Docker image and run a container, and then perform a simple experiment to verify things work.

2.1 Environment Setup

First, we must load the Docker image and start a container. You can do it in the following way:

```
docker load < iea.tar.gz
docker run -it iea
cd $IEA_HOME
```

Now, we are in a Debian container, ready to run experiments. For convenience, some tools like `vim` and `curl` are available. All instructions should be performed within this container unless otherwise specified.

2.2 Simple Experiment

We will start with two scripts `$IEA_HOME/bench/run-aa` and `$IEA_HOME/bench/run-vf` to verify whether Iea is successfully set up:

(1) `cd $IEA_HOME/bench/graphtexts/aa`
(2) `aa -sccpocr avrora.g`
(3) `cd $IEA_HOME/bench/graphtexts/vf`
(4) `vf -sccpocr xz.g`

This will perform 1 round of alias analysis on `avrora.g` and 1 round of value-flow analysis on `xz.g` using Iea solver. If it prints messages on the screen like the following two graphs,

```
GraphSimpTime   0.192
#PEdges 11298
#Nodes  10650
#Edges  19294
VmrssInGB       0.0404701
AnalysisTime    3.397
#Iterations     1
#SumEdges       1656740
#Checks 1333344
```

```
OfflineSCCTime  0.275
GraphSimpTime   0.873
OfflineGFTime   0.593
OnlineSCCTime   0.41
#PEdges 6956
#OrigNodes      31267
```

it means that Iea is successfully set up.

3 STEP-BY-STEP INSTRUCTIONS

This section details usage of the benchmarking script and how to reproduce our results.

3.1 Benchmarks and CFL-reachability Solvers

For context-sensitive value-flow analysis, we use 10 C/C++ programs in the SPEC 2017 benchmark suite. They are sorted from the smallest to the largest base on their size as follows:

`xz nab leela x264 cactus povray imagick parest perlbench omnetpp`

For field-sensitive alias analysis, we use 10 Java programs from the DaCapo benchmark suite. They are sorted from the smallest to the largest base on their size as follows:

`avrora biojava h2o batik fop derby jme cassandra pmd lucene`

The SVFGs of the SPEC 2017 C/C++ benchmarks are drawn from the bitcodes compiled using Clang-14.0.0 and linked by wllvm 1. The PEGs of the DaCapo Java benchmarks are generated by converting the Java bytecode using Soot.

We have implemented **4 CFL-reachability solvers**, i.e.,

(1) `std` - the standard CFL-reachability solver, which is our baseline.
(2) `iea` - Iea solver (Section 4.2.2 of our paper).
(3) `pocr` - Pocr solver with optimizations.
(4) `ieaocr` - Iea-Ocrsolver (Section 5.3 of our paper).

3.2 Benchmarking Script

The basic benchmarking scripts `run-aa.sh` and `text-vf.sh` are for alias analysis and value-flow analysis respectively. They are to be used as follows:

```
usage:  ./run-aa.sh SOLVER_TYPE INPUT_FILES
        ./run-vf.sh SOLVER_TYPE INPUT_FILES
```

- `SOLVER_TYPE` refers to the type of chosen CFL-reachability solver, it must be one of the four CFL-reachability solvers listed above.
- `INPUT_FILES` refers to the benchmark(s) to be analyzed. You can input one or more of the 10 benchmarks listed above and the scripts will print the results of each of them.

The benchmarking script `shortTest.sh` is for performing both alias analysis and value-flow analysis using all the 4 CFL-reachability solvers on one or more benchmarks you selected from what we displayed above. It is to be used as follows:

```
usage:  ./shortTest-aa.sh INPUT_FILES
usage:  ./shortTest-vf.sh INPUT_FILES
```

where `INPUT_FILES` refers to one or more benchmarks selected from what we displayed above.

For example,

```
        ./shortTest-vf.sh xz nab leela
```

will run value-flow analysis on the three benchmarks using all the 4 CFL-reachability solvers listed in Section 3.1.

This is expected to complete within less than 20 minutes on a platform with a Quad-core Intel Xeon 2.10 GHz CPU and 16 GB memory.

3.3 Reproducing Our Results

The benchmarking script `fullTest.sh` is to perform the full experiment of our paper. Our experimental results can be reproduced by simply running

```
    ./fullTest.sh
```

That is, we perform both alias and value-flow analysis for all benchmarks by running all the 4 CFL-reachability solvers (listed in Section 3.1) for two clients value-flow analysis and alias analysis one by one.

Note:

- The analyses of the two largest programs may not be successful on platforms with memory less than 64 GB.
- When running `./fullTest.sh`, you can terminate it using Ctrl+C. If there are failed cased use `reTest.sh` to only run failed cased.
- Use `tab1Gen.sh`, `tab2Gen.sh` and `tab3Gen.sh` in `$IEA_HOME/bench/` to generate complete resulting tables. And you may see the results like Figure 1.

3.4 Running Your Own Analysis

In addition to analyzing bitcodes, you can also run alias and value-flow analysis on graphs written in text in this artifact, and compare the performances of the 4 CFL-reachability solvers. The input graph should be in a format that each line denotes an edge in the following form:

```
    [EDGE SOURCE]   [EDGE DESTINATION]  [EDGE LABEL]    [LABEL INDEX]
```

The elements of each line are separated by tab characters (i.e., `'\t'`). Specifically, `[EDGE SOURCE]`, `[EDGE DESTINATION]` and `[LABEL INDEX]` are integers. For value-flow analysis, the value of `[EDGE LABEL]` should be one of `a`, `call` and `ret`, corresponding to Figure 9 of our paper. For alias analysis,

Table 1 Benchmark info for value-flow analysis.

#Node and #Edge refer to the total number of nodes and edges in the original graphs, respectively. Node% and Edge% denote the percentage of reduction in nodes and edges achieved through offline graph simplification techniques. OSCC stands for offline cycle elimination, while OVS represents offline variable substitution. MemoryUsage measures the amount of memory utilized during the solving process by two different methods, denoted as Std and Pocr, measured in gigabytes (GB).

Bench.	Size/MB	SVFG		OSCC+OCS		Memory Usage/GB		Runtime	
		#Node	#Edge	Node%	Edge%	STD	POCR	STD	POCR
1.xz	1.24	49395(13811)	62955(22523)	72.0396	64.2236	0.011055	0.0555	0.505	0.207
2.nab	1.41	55652(11772)	72366(19199)	78.8471	73.4695	0.0968513	0.522465	7.088	2.956
3.leela	2.93	64466(20268)	89081(39212)	68.5601	55.9816	0.0217209	0.150188	1.972	0.72
4.x264	4.68	207064(62540)	340217(156531)	69.7967	53.9908	0.545181	3.08361	112.491	19.134
5.cactus	5.88	544480(175008)	1007989(493365)	67.8577	51.0545	1.2729	13.4819	1631.28	128.911
6.povray	7.38	537775(163940)	1041687(506714)	69.5151	51.3564	1.83455	13.9192	1795.78	127.629
7.imagick	13.68	574089(144948)	842509(293549)	74.7516	65.1577	0.461929	4.12052	604.532	50.701
8.parest	16.20	299718(108286)	407343(197568)	63.8707	51.4983	0.0791245	0.226982	3.448	1.058
9.perlbench	18.69	697744(303738)	1662445(1086140)	56.4685	34.6661	5.68657	-	13440	-
10.omnetpp	21.81	664358(204182)	1857831(1214208)	69.2662	34.6437	1.04406	1.87209	309.337	12.423
Average				69.10	53.60	1.105	3.743	1790.643	34.374

Fig. 1

Table 2. Benchmark info for alias analysis. The meanings of column headings are the same as Table 1.

Bench.	Size/MB	PEG		OSCC+OCS		Memory Usage/GB		Runtime	
		#Node	#Edge	Node%	Edge%	STD	POCR	STD	POCR
1.avrora	1.71	80981(10650)	91532(19294)	86.8487	78.9210	0.0566559	0.0405655	8.388	0.289
2.biojava	3.54	96634(-)	105421(-)	-	-	-	-	-	-
3.h2o	4.13	21930(16462)	25012(33702)	24.9338	-34.7433	0.119743	0.074604	24.452	0.624
4.batik	13.61	45752(41913)	85561(95112)	8.3908	-11.1627	1.12092	0.973354	412.314	13.794
5.fop	16.67	78991(45359)	101972(108240)	42.5770	-6.1467	2.31912	2.42524	1306.44	33.555
6.derby	19.94	78764(51845)	140235(123874)	34.1767	11.6668	1.33115	1.05658	2540.72	15.59
7.jme	45.13	93861(-)	130546(-)	-	-	-	-	-	-
8.cassandra	49.51	716674(-)	95861(-)	-	-	-	16.3382	-	296.523
9.pmd	56.85	628244(-)	795621(-)	-	-	-	57.7794	-	1012.28
10.lucene	66.10	769161(113526)	562458(271136)	85.2402	51.7944	7.71815	7.88305	27817.5	146.858
Average				28.22	9.03	1.267	8.657	3210.981	151.951

Fig. 2

the value of `[EDGE LABEL]` should be one of `a`, `abar`, `d`, `dbar`, `f` and `fbar`, corresponding to Figure 10 of our paper.

The commands for performing alias analysis and value-flow analysis are listed as follows,

```
aa -SOLVER_TYPE xxx
vf -SOLVER_TYPE yyy
```

3.5 Source Code

The important source code files (among many others) are available in the following files:

- `svf/include/CFL` are the main folders contains CFL related module.
- `svf/include/CFL/CFLSolver.h` contains core solver algorithm.
- `svf/include/Graphs/CFLGraph.h` are the memory representation of graph.
- `svf-llvm/tools/CFL` are the tools module.

Plan. We plan to integrate our work to the latest version of open-source tool SVF (https://github.com/SVF-tools/SVF) Docker image on this can be seen at https://hub.docker.com/repository/docker/talbenxu/iea/general.