On-Demand Strong Update Analysis via Value-Flow Refinement

Artifact Track

Yulei Sui Jingling Xue

School of Computer Science and Engineering, UNSW Australia

ysui@cse.unsw.edu.au, jingling@cse.unsw.edu.au

1. Introduction

SUPA is a new demand-driven **S**trong **UP**date **A**nalysis that computes points-to information on-demand via value-flow refinement. We formulate SUPA by solving a graph-reachability problem on a value-flow graph representation of the program, so that strong updates are performed where needed with imprecisely pre-computed value-flows being refined away, as long as the total analysis budget is not exhausted. SUPA facilitates efficiency and precision tradeoffs by allowing different pointer analyses to be applied in a hybrid multi-stage analysis framework.

The artifact includes a full implementation of SUPA, some benchmarks and scripts to reproduce the data in our FSE 2016 paper.

2. Description – Scorecard

We describe our artifact based on the following three dimensions (i.e., insightful, useful, and usable):

2.1 Insightful

As a fundamental program analysis, pointer analysis paves the way for program optimisations and software bug detections, such as detecting memory leaks, uninitialised variables, security vulnerabilities, and tainted information flow. The more precisely the points-to information is resolved, the more effective static analysis will potentially be.

In the artifact, we provide a hybrid analysis framework to find a right balance between efficiency and precision via a demand-driven strong update analysis. Our tool can achieve almost the same precision as whole-program analysis by consuming about 0.19 seconds and 36KB of memory per client query for uninitialised variable detection. As also acknowledged by our FSE paper reviewers, the proposed approach is "novel" and "solid" on a "very timely topic that demand-driven analyses are dearly needed".

2.2 Useful

SUPA is built on top of SVF (`http://unsw-corg.github.io/SVF/`), a static value-flow analysis framework operating on LLVM-IR. SUPA will be integrated into SVF based on the latest release of LLVM (version 3.8.0).

The artifact provides a platform for evaluating both whole-program and demand-driven pointer analyses based on the popular compiler LLVM. Our tool, which is under active development and maintenance, has already benefited from a large and rapidly growing user base, including researchers and developers from both academia and industries who are working in the broad area of programming languages and software engineering.

2.3 Usable

We have provided a detailed checklist of our package (Section 3), including online guidelines (`http://www.cse.unsw.edu.au/~corg/supa`) to reproduce the results in our paper, and Wiki documents (`https://github.com/unsw-corg/SVF/wiki`) to understand its internal working and develop new analyses in our framework.

To help users reproduce the data in the paper, we have also provided a virtual machine image (Section 3), which contains the SUPA releases, together with scripts and benchmarks used in our experimental evaluation, as described in the paper.

3. Artifact Package

You may find the package and all the instructions on how to use SUPA via `http://www.cse.unsw.edu.au/~corg/supa`.

A brief checklist:

- `index.html`: the detailed instructions for reproducing the experimental results in the paper.
- `SUPA.ova`: a virtual image file ($\sim$5GB) containing installed Ubuntu OS and SUPA project (`http://corg-pluto.cse.unsw.edu.au/supa/SUPA.ova`).
- SUPA source code developed on top of the `SVF` framework: `http://unsw-corg.github.io/SVF`.
- Scripts for reproducing all the data in the paper, including:
  - `./run.sh`,
  - `./figure_9.sh`,
  - `./figure_10.sh`,

  - `./figure_11.sh`,
  - `./table_2.sh`,
  - `./table_3.sh`.
- Micro-benchmarks to validate pointer analysis results.

Platform: All the results related to analysis times and memory usage in our paper are obtained on a 3.7G Hz Intel Xeon 8-core CPU and 64 GB memory.The OS in the virtual machine image is Ubuntu 12.04. A user account has been created with both its username and password as "pta".

To run SUPA, you are advised to allocate at least 16GB memory to the virtual machine. The whole-program sparse flow analysis, denoted SFS in the paper, requires more memory, as a lower memory budget may force OS to kill the running process when it is used to analyse some large programs, e.g., vim, gdb and emacs. Finally, a VirtualBox with version 4.1.12 or newer is required to run the image.

License: GPLv3 (`www.gnu.org/licenses/gpl-3.0.en.html`)

4. Project Layout

SUPA project is located under the "pta" directory:

```
pta                    //  Top  directory
 |-include             //  The executables
 |   |-DDA             //  SUPA header  files
 |   |-MSSA            //  Memory SSA and VFG
 |   |-Util            //  Analysis  utils
 |   |-MemoryModel    // Abstract  Memory Modeling
 |-lib                 //  The executables
 |   |-DDA             //  SUPA
 |   |-MSSA            //  Memory SSA and VFG
 |   |-Util            //  Analysis
 |   |-MemoryModel    // Abstract  Memory Modeling
 |-Release+Asserts     //  The executables
 |   |-bin
 |   |   |-dvf         //  SUPA executable
 |   |-lib
 |SUPA-exp
 |   |-*.orig          //  Benchmarks
 |   |-*.sh, *.pl      //  Scripts  for  running  tests
 |   |-supa-results    //  Experimental  data  in  the  paper
 |- tests              //  Micro-benchmark suite
 |-setup.sh            //  Setup  script
```

5. Quick Guidelines

To run the experiments as we did for in our paper, open a terminal and do:

- **cd /home/pta/pta/** # Go to the SUPA project directory, denoted as $SUPAHOME
- **. ./setup.sh** # Set up environment variables (please note that there is a white space between the two dots)
- **cd SUPA-exp** # Go to the experiment directory

To run the three analyses for all 12 benchmarks, execute the following scripts:

- **./run.sh DFS** # Run SUPA-FS analysis only
- **./run.sh CXT** # Run SUPA-FSCS analysis
- **./run.sh SFS** # Run whole-program sparse flow-sensitive analysis

Please note that all the results related to analysis times and memory usage in our paper are obtained on a machine with 3.7G Hz Intel Xeon 8-core CPU and 64 GB memory. On this platform, obtaining the results for SUPA may take about 30 mins with a budget of 1000, and obtaining the results for SFS may take more than five hours (especially for large programs, such as bash, vim and emacs).

Initially, the users are advised to analyse a few benchmarks with small budgets using the default configuration files 'budget' and 'benchmarks'. To analyse all the benchmarks, please use another configuration file containing all the benchmarks (and remember to re-run everything if the configuration files are changed):

After the analyses are all *fully* finished, you can collect data for tables and figures using scripts in the same folder:

- **./figure_9.sh** # Data in Figure 9
- **./figure_10.sh** # Data in Figure 10
- **./figure_11.sh** # Data in Figure 11
- **./table_2.sh** # Data in Table 2
- **./table_3.sh** # Data in Table 3

For comparison purposes, we have also provided the experimental data presented in the paper under "$SUPAHOME/SUPA-exp/supa-fse/".

To reproduce the results shown in tables and figures with the provided data, issue the following commands:

- **./figure_9.sh supa-fse** # Data in Figure 9
- **./figure_10.sh supa-fse** # Data in Figure 10
- **./figure_11.sh supa-fse** # Data in Figure 11
- **./table_2.sh supa-fse** # Data in Table 2
- **./table_3.sh supa-fse** # Data in Table 3

6. More Experiments and Developer Guide

Please refer to

- The website of SUPA (`http://www.cse.unsw.edu.au/~corg/supa`)
- The Wiki site of our SVF framework (`http://unsw-corg.github.io/SVF`).