CS 6340: Software Analysis and Test, Fall 2023
SVF Project

1 Project Description

In this project, you will get familiar with the SVF framework, and build a reachability analysis tool working on LLVM IR. SVF is a static tool that enables scalable and precise interprocedural dependence analysis on LLVM IRs. Your submitted analysis tool should read LLVM IR files and decide (statically) whether the function call `src`() reaches the function call `sink`().

1.1 Task Description

An interprocedural control-flow graph (ICFG) describes the control flow of a target program. The ICFG represents the control instructions from the program entry node to the program exit node and provides multiple control flows among the whole program. Analysis of an ICFG can be used to detect path reachability between two nodes.

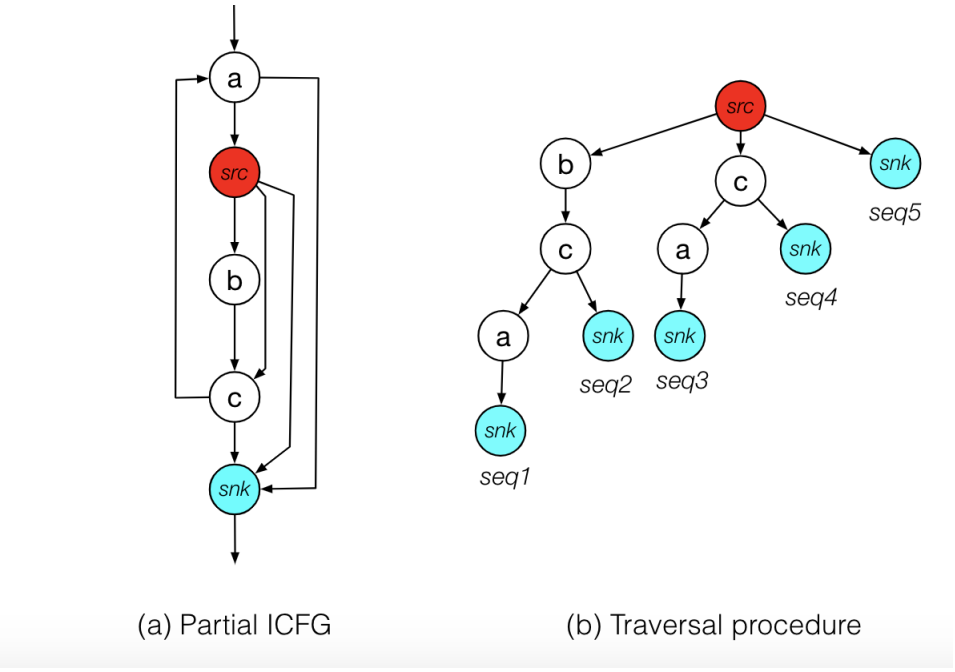

Figure 1: Example

Your program should perform the previously mentioned reachability analysis. Some caveats:

1. The analysis should be **Path-Insensitive & Context-Sensitive**. No data flow analysis is necessary. As such, paths that are infeasible can appear in your output:

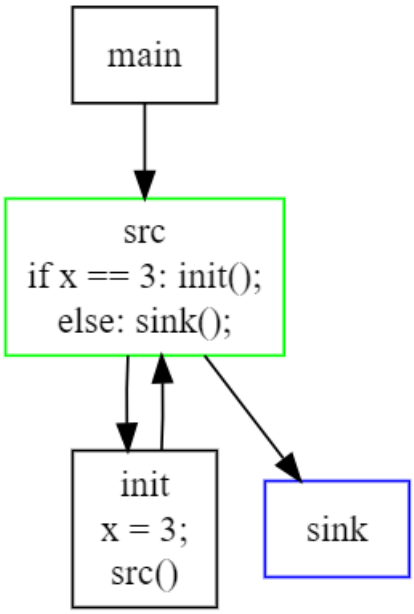

Figure 2: Interprocedural Case

   (a) $src -- > sink$ (infeasible)

   (b) $src -- > init -- > src -- > sink$

   (c) $Cycle[src -- > init -- > src] -- > sink$ (infeasible)

   (d) $src -- > Cycle[init -- > src] -- > sink$ (infeasible)

2. You do not need to consider recursive function call cycles in this project.

The ICFG can be generated by the SVF framework; SVF also provides the functionality to identify instructions represented by the nodes in the ICFG.

One goal in this project is also to learn how to utilize a real-world static analysis framework to implement analysis tools. Please start early to get familiar with the SVF framework in this project! Reference for the SVF framework is available at https://github.com/SVF-tools/SVF/wiki.

1.2 Project Setup

1.2.1 Docker

To avoid any configuration issues, we will use Docker in this project. We provide a guide to use Docker in your project, and our grading environment will be the same as described in the guide. Here are the instructions to set up, build and test your project.

First you need to install Docker on your local machine. Here is the link to help to get started with Docker: https://www.docker.com/get-started.

Once you have Docker installed on your machine, you can pull the provided Docker image from Docker Hub and then work on it. Here is a guide of the entire process. If you want to know more about Docker please consult the official documentation.

1. First you can pull the image from Docker Hub.

   ```
   docker pull mdavis438/svfproject:latest
   ```

2. Then you can start a Docker container from the provided Docker image using the following command.

   ```
   docker run -it --name svfproject mdavis438/svfproject:latest /bin/bash
   ```

   If you want to come back later you can exit the bash session and start the stopped container again using the following command.

   ```
   docker start -ai svfproject
   ```

Please keep in mind that docker containers will not save changes when closed! Either:

1. **Save the container to a docker image**
2. **Mount the project directory to your local computer**
   (add `-v host/directory/path:/home/project/src` to your docker run command)
3. **Save the file copy to your local machine before closing the container**

1.2.2 VSCode (optional)

This section provides a guide for if you want to use VScode dev container connections.

1. Install VSCode Editor: https://code.visualstudio.com/download
2. Install Remote Development Extension in VSCode
3. This alternative of the docker start command will start the container in the background without an interactive terminal:
   `docker start svfproject`
4. Attach to the docker container.

   Open the Remote Explorer view from the left panel. Select "Dev Containers" in the dropdown at the top of the sidebar. This should list all of the docker containers available.

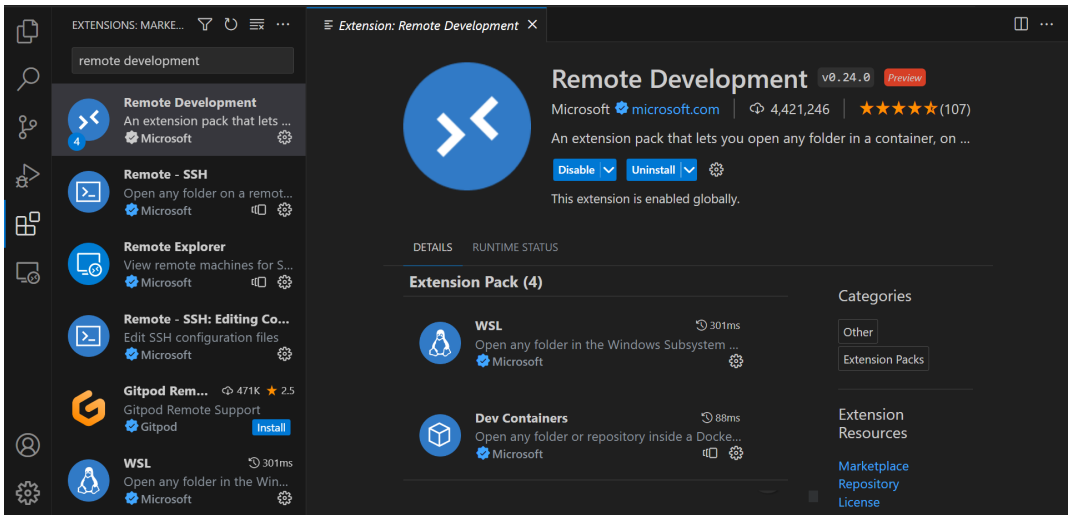

Figure 3: Remote Development Extension in VSCode

Choose your project container ('svfproject'). VSCode will take some time to setup the connection and then open the container environment.

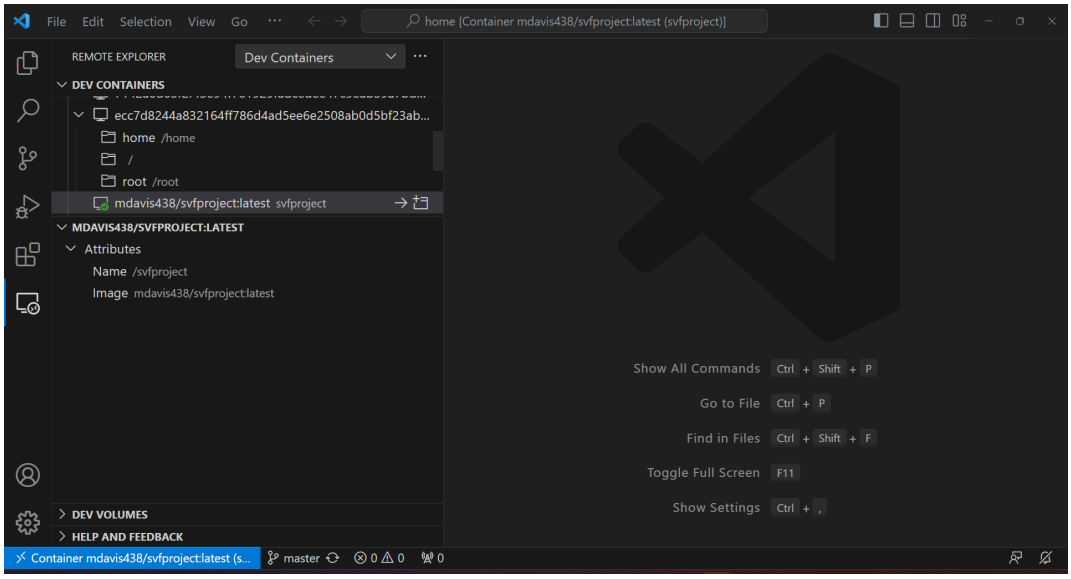

Figure 4: Dev Containers in VSCode

More info on dev containers can be found here. If you're curious you can follow this tutorial but replace the example container instructions with our docker container.

1.2.3 Project Setup

1. To complete this project, you only need to modify the following file inside the container.

`~/project/src/project.cpp`

2. After completing the code in this file you can build the project by first going to the directory "`~/project/`" and then running the following commands

```
source ./env.sh
./build.sh
```

3. This will generate the executable `~/project/bin/svf-project`. The project can be run using the command

```
./test.sh [path/to/test.bc]
```

and will generate the `out.txt` output file

2 Provided Code

The only provided code is in the Docker container on Docker Hub, so there is no project directory provided. In the Docker container we have only provided the framework to build the project based on the dependence of SVF libraries. There will only be a placeholder `project.cpp` file. In this file, you need to use the SVF framework to generate the ICFG graphs of the input LLVM IR file. And you need to use SVF to identify the nodes representing the `src`() function call and `sink`() function call. Then your project should perform the reachability analysis and print the result to `stdout`.

3 Grading

There are in total 100 points for this project.

3.1 Correct Implementations (60 Points)

Several test cases are given to you to test the correctness of your project under the `~/project/test_cases` directory.

You can run the script `test.sh` to check the correctness of your project for these test cases. For grading, there will be hidden test cases for the project.

For each input test cases, your program is expected to print out the following outputs:

- **First line:** The result whether the `src`() function call can reach the `sink`() function call (40 Points). If it is reachable, your program should print "Reachable", otherwise print "Unreachable". Note that please print the EXACT text of "Reachable" or "Unreachable". For example, "UnReachable" is not an acceptable output. If your program provides correct answers to the reachability results, you can get 40 Points.

- **Following lines:** If there are n traversal paths between the `src` and `sink`, your project should also print n more lines. Each line is a path from `src`() to `sink`() (20 Points). The path should have the following format, supposing that node 1 is the `src` node and node 5 is the `sink` node: $1 -- > 2 -- > 3 -- > 4 -- > 5$. In the instance that the path or a portion is cyclic, annotate the cyclic region in the following format: $1 -- > Cycle[2 -- > 3 -- > 4] -- > 5$. The exit node from the cycle must be correct, regardless if this duplicates nodes i.e. $1 -- > Cycle[2 -- > 3 -- > 4 -- > 2] -- > 5$. Correct program traversal paths will award 20 Points.

3.2 Performance Evaluation (10 Points)

If your program can finish running each test case in 30 seconds, you will get 10 Points.

3.3 Design (30 Points)

In your final report, please briefly describe the following:

1. Details about how you generate the ICFG from the input LLVM IR.
2. Details about how you perform the reachability analysis.
3. Any known outstanding bugs or deficiencies that you were unable to resolve before the project submission.

4 Submission

To facilitate the grading, you should not modify other files in the project directory! On Canvas, submit a single ZIP file that contains:

- The `project.cpp` file
- The `design.pdf` file described in Section 3.3.
- Any test cases you added. (This will not be graded.)

We will only use the `project.cpp` file to test the correctness of your implementation. If you modify other files, even if your code runs correctly on your local machine, you will still get points deducted if it cannot run properly in our grading environment.

5 Collaboration

We will award identical grades to each member of a given project team, unless members of the team directly register a formal complaint. We assume that the work submitted by each team is their work solely. Any questions about design and implementation should conducted solely with the instructors through office hours, email, etc. Under no condition is it acceptable to use code written by another team, or obtained from any other source. As part of the standard grading

process, each submitted solution will automatically be checked for similarity with other submitted solutions and with other known implementations.