CS 4240: Compilers, Project 3 (SVF), Spring 2024

Assigned: March 25, 2024, 5% of course grade

Due on Gradescope by 11:59pm on April 17, 2024

1 Project Description

In this project, you will get familiar with the SVF framework, and use it to build a simple reachability analysis tool working on LLVM IR (encoded as the LLVM Bitcode File Format). Your tool should decide whether there exist paths from the **function call** `src()` to the **function call** `sink()` in the inter-procedural control-flow graph (ICFG), which can be generated by SVF.

1.1 Detailed Task Description

Inter-procedural control-flow graph (ICFG) is more general than normal control-flow graph (CFG) in that it also describes function call relations. Conceptually, an ICFG is the result of connecting the CFGs for individual functions via caller-callee relations.

Consider the `src` and `sink` nodes in part of ICFG in Figure 1(a); it is a directed graph with cycles. The `src` node is an instruction that contains the `src()` function call. The `sink` node is an instruction that contains the `sink()` function call. It is guaranteed that in each test case there will be exactly one `src()` function call and exactly one `sink()` function call. In Figure 1(b), five traversing sequences are provided from `src` to `sink`. In this project, you will count the number of traversing sequences between the two vertices `src` and `sink`, where each node in each path of the sequences should only be visited once (paths containing cycles should not be counted).

1.2 Project Setup

Please use the latest stable release version SVF-2.9. We will also use this version to grade your submission. You can build SVF following the provided instructions. Note that you might need additional dependencies like `xz-utils`. Please modify the example analysis tool located at `svf-llvm/tools/Example/svf-ex.cpp` provided by SVF to achieve this project's goal. Building SVF will automatically build that tool. The SVF doxygen document is a good place to find SVF class / function names. You can also find the names using IDE features, such as IntelliSense in Visual Studio Code.

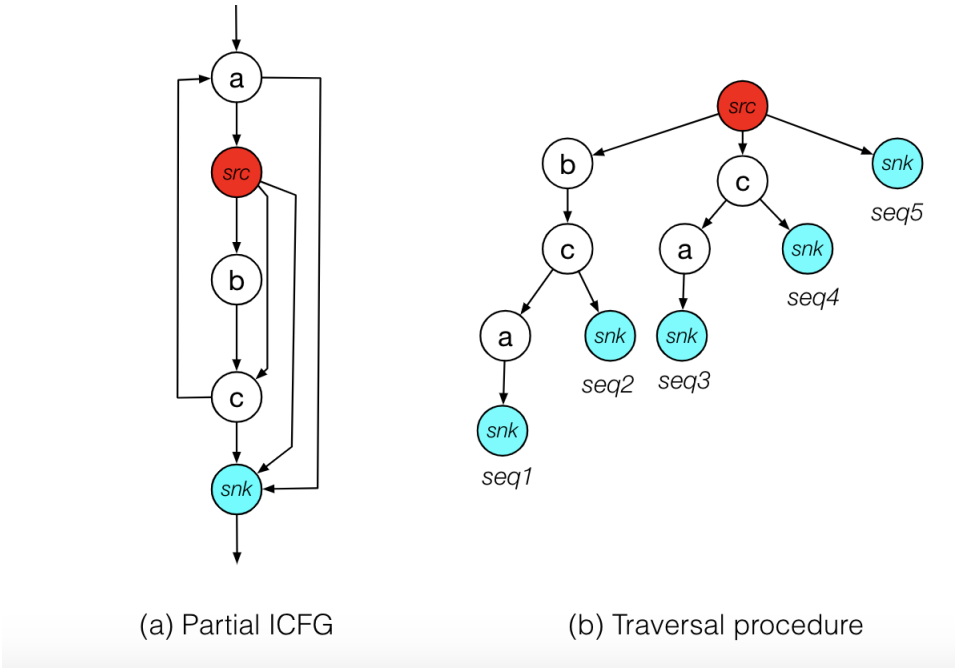

Figure 1: ICFG example.

2 Provided Code

We provide 2 public test cases (C programs) in Figure 2, and there are 5 hidden test cases.

Before testing your tool, please compile the C test cases to LLVM IR bitcode files so your SVF-based tool can read them. We recommend you to use the exact Clang/LLVM version (14.0.0) as SVF-2.9's dependency to do this: after building SVF, you can find it at `llvm-14.0.0.obj/bin/clang-14`. The function names `src` and `sink` will be preserved in the bitcode files (and hence in the ICFG generated by SVF).

3 Grading

3.1 Correct Implementation (70 Points)

For each test case, your program is expected to print out the following outputs within 10 seconds; a timeout will result in zero point.

- **First line:** The result whether the `src()` function call can reach the `sink()` function call (5 points). Print `"Reachable"` or `"Unreachable"` (excluding quotes).
- **Second line:** A natural number n denoting the number of traversal paths between `src()` and `sink()` (5 points). 0 should be printed when there is no such path.

```c
void src() {}
void sink() {}

int cond() { return 1; }

int main(void) {
    src();
    if (cond()) {} else {}
    sink();
    return 0;
}
```

(a) This test case has two reachable paths: (1) `src() -> cond() -> "the if branch" -> sink()`, and (2) `src() -> cond() -> "the else branch" -> sink()`.

```c
void src() {}
void sink() {}

void f() {}
void g() { f(); sink(); }

int main(void) {
    src();
    f();
    return 0;
}
```

(b) This test has one reachable path: `src() -> "call into f in main" -> "return from f in g" -> sink()`. This cannot be a real execution path since you cannot call into `f` from one call-site and return to another call-site, but ICFG links the two call-sites to the same `f`, so your traversal will get such a spurious path. **For this project, you are expected to count these spurious paths.** Eliminating such paths is an important research topic.

Figure 2: The two public test cases.

3.2 Design (30 Points)

In your final report, please briefly describe the SVF API you use to generate ICFGs, the details about your graph traversal algorithm, and any known bugs or deficiencies.

4 Submission

On Gradescope, submit a single ZIP file that contains:

- The `svf-ex.cpp` file;
- The `design.pdf` file described in Section 3.2.

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 clarification question about the project handout should be posted on the course's public Piazza message board. Any non-obvious discussion or questions about design and implementation should be either posted on the course's Piazza message boards privately for the instructors or presented in person during office hours. If the instructors determine that parts of

the discussion are appropriate for the entire class, then they will forward selections. 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.