Lab Assignment 7

Course: CS202 Software Tools and Techniques for CSE
Lab Topic: Reaching Definitions Analyzer for C Programs
Date: 15th September 2025

Objective

The purpose of this lab is to implement program analysis techniques taught. Students will construct Control Flow Graphs (CFGs) and perform Reaching Definitions Analysis on non-trivial programs. This assignment is designed to make students go beyond toy examples and engage in structured program analysis that a software tool performs behind the scenes.

Learning Outcomes

By the end of this lab, students will be able to:

- ✓ Construct and visualize CFGs for small-to-medium programs.
- ✓ Implement **Reaching Definitions Analysis** using data-flow equations.

Lab Requirements

- ➢ Any Operating System (Windows, Linux, MacOS, etc.).
- ➢ Programming Language: Python 3.10 or later
- ➢ Read Lecture 7 slides carefully.
- ➢ Required Tools:
  - **Graphviz** (for CFG visualization)
  - **Matplotlib** (for plotting graphs)

Lab Activities:

1. **Program Corpus Selection:**
   Choose **three** C programs (200-300 LOC each). The source files should be standalone source files for ease of analyses. This means each program is exactly one `.c` file with a `main()` function). For simplification, you may consider only programs with a single function and perform intraprocedural analysis instead of interprocedural analysis.
   - ➢ Programs should include:
     - Conditionals (if/else).
     - Loops (while, for).
     - Multiple variables with reassignments.

   List the chosen programs with a **short justification** (why selected).
2. **CFG (Control Flow Graph) Construction:**
   - ➢ Rules to find **leaders** (the first instructions of basic blocks):
     - The **first instruction** of the program is a leader.
     - Any instruction that is the **target of a branch/jump/loop** is a leader.
     - Any instruction that comes **immediately after a branch/jump/loop** is also a leader.
     - **Example C code:**

```c
int main() {
    int x = 0;
    int y = 5;
```

Note: Please reach out to the TAs for any queries/issues.

```
    if (y > 0) {
        x = y + 1;
    }  else {
        x = y - 1;
     }
     printf("%d", x);
     return 0;
}
```

1. First instruction (`int x = 0;`) → **leader**.
2. `if (y > 0)` → **leader**.
3. `x = y + 1;` (target of if true) → **leader**.
4. `x = y - 1;` (target of else) → **leader**.
5. `printf("%d", x);` (immediately after if/else) → **leader**.

➢ Identify Basic Blocks:

- A **basic block** is a maximal sequence of instructions that:
  Has **exactly one entry point.**
  Has **exactly one exit point.**

  Grouping into basic blocks (Example based on the above C code):
  1. B0: `int x = 0; int y = 5;`
  2. B1: `if (y > 0)`
  3. B2: `x = y + 1;`
  4. B3: `x = y - 1;`
  5. B4: `printf("%d", x); return 0;`

➢ Construct the CFG:

- Each basic block corresponds to one node in the CFG.
- Add **edges** according to control flow:
  Sequential flow → edge from block to the next.
  If/else → branch from condition block to each branch body.
  While/for loop → edge from condition block to loop body and to exit; edge from loop body back to condition (back edge).
- Adding Edges (Example based on the above C code):
  1. Sequential: B0 → B1
  2. If condition: B1 → B2 (true branch), B1 → B3 (false branch)
  3. After if/else: B2 → B4, B3 → B4
- Use **Graphviz** to draw the CFG. To do this, first store CFG information in `.dot` format (a plain text graph description language used by **Graphviz**).
- Each node in the .dot file should be labeled with the actual code of the basic block. Then, render the `.dot` file into an image (PNG/PDF) using the `dot` command.
- For example,
  For each **basic block**, create a node in DOT with a label showing its code.
  `B0 [label="B0:\nint x=0;\nint y=5;"];`

  For each **control flow edge**, add a directed edge in DOT.

Note: Please reach out to the TAs for any queries/issues.

```
B1 -> B2 [label = "true"];
B1 -> B3 [label = "false"];
```

Save the description with `.dot` extension and use **Graphviz** to generate the corresponding CFG.

- Provide **CFG diagrams** for each program.

3. **Compute Cyclometric Metric Complexity Metrics:**
   - For each program, extend your **CFG construction tool** to automatically calculate the following metrics:
     - **N (Number of Nodes)** = Number of basic blocks.
     - **E (Number of Edges)** = Number of control flow edges.

       Note: These metrics must be computed **automatically by your tool**, not manually.
     - Cyclomatic Complexity (CC): ***E – N + 2***

       (This is a well-known measure of program complexity. Higher CC → more complex control flow.)
     - Provide a **metrics table**:

Program No.	No. Of Nodes (N)	No. Of Edges (E)	Cyclomatic Complexity (CC)
...	...	...	...
...	...	...	...

4. **Reaching Definitions Analysis:**
   - Identify Definitions:
     - A **definition** is any assignment statement (e.g., `x = 5;`)
     - Give each definition a unique ID (D1, D2, D3, ...).
     - Write down the mapping between each definition ID and the corresponding code line.
   - Compute **gen[B]** and **kill[B]** for Each Basic Block B:
     - **gen[B] (generated definitions):**

       All definitions that are created inside block B.
     - **kill[B] (killed definitions)**:

       All *other* definitions of the same variables that exist outside block B.
   - Apply dataflow equations:
     - For each basic block B, compute:

       in[B] = ∪ out[P] where P = predecessors of B.

       out[B] = gen[B] ∪ (in[B] – kill[B]). Where,
       - The set **in[B]**: all definitions that can reach block B from its predecessors.
       - The set **out[B]**: definitions created in B plus the ones from predecessors that are not overwritten in B.
   - Iterative Computation Until Convergence:
     - Initialize all sets (**in[B]**, **out[B]**) as empty.
     - Apply the equations repeatedly.
     - Stop when the sets stop changing (this is called **convergence**).
   - Produce results in a **tabular format for each iteration**:

Note: Please reach out to the TAs for any queries/issues.

Basic-Block	gen[B]	kill[B]	in[B]	out[B]
…	…	…	…	
…	…	…	…	

- ➢ Interpretation of Results:
  Based on in[B] and out[B] sets and explain:
  Which variables may have **multiple possible reaching definitions** at the same program point?

Resources

- Lecture 7 Slides
- https://en.wikipedia.org/wiki/Data-flow_analysis
- https://graphviz.org/
- https://pygraphviz.github.io/
- https://dl.acm.org/doi/pdf/10.1109/ICSE-SEIP58684.2023.00025

Note: Please reach out to the TAs for any queries/issues.