CS202 Lab 7 Report: Reaching Definitions Analyzer for C Programs

Md Sibtain Raza (22110148) | Github Link

15 October 2025

1 Objective

The primary objective of this lab was to design, implement, and test a program analysis tool capable of performing Reaching Definitions Analysis on C programs. This involved two main tasks: first, constructing and visualizing a Control Flow Graph (CFG) for a given C program, and second, implementing an iterative dataflow analysis algorithm to compute the 'in' and 'out' sets for each basic block until convergence.

2 Methodology

2.1 Environment

Operating System: Windows, with Python 3.10+ and Graphviz installed so the dot executable is available on PATH for rendering DOT to PNG as mandated in the lab. A Python virtual environment was created and activated. All necessary tools and libraries were installed using pip.

```
(.venv) PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab7> python --version
Python 3.13.7
(.venv) PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab7> dot -V
dot - graphviz version 14.0.1 (20251006.0113)
(.venv) PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab7>
```

2.2 Program Corpus Selection

Three standalone C programs were selected to serve as the corpus for this analysis. The programs were chosen based on the lab requirements (200-300 lines of code) and the presence of varied control flow structures (conditionals, loops) and frequent variable reassignments, making them suitable candidates for dataflow analysis.

1. `program1.c` **(Matrix Operations):** A menu-driven program that performs matrix addition, subtraction, multiplication, and transpose operations. Its complexity arises from multiple nested loops and a main 'while' loop for the menu.↗
2. `program2.c` **(Number Analyzer):** A tool that provides various numerical functions, including prime checking, factorial calculation, and Fibonacci series generation. It contains a mix of loops and conditional branches.↗
3. `program3.c` **(Tic-Tac-Toe Game):** A fully playable Tic-Tac-Toe game with a scoreboard. This program features nested loops for board management and complex conditional logic for determining game state (win, draw, continue).↗

2.3 Analyzer Implementation

A command-line tool named `cs202_lab7.py` was developed in Python to automate the entire analysis pipeline. The script's functionality is broken down as follows:

```python
import os, sys, json, subprocess, re
from collections import defaultdict
from pycparser import c_parser, c_ast
from pycparser.c_generator import CGenerator

# ---------- Utilities ----------
def read_text(p):
    return open(p, 'r', encoding='utf-8', errors='ignore').read()

def write_text(p, s):
    os.makedirs(os.path.dirname(p), exist_ok=True)
    with open(p, 'w', encoding='utf-8') as f: f.write(s)

def strip_preprocessor(src):
    # Remove preprocessor directives
    no_preproc = "\n".join([ln for ln in src.splitlines() if not ln.lstrip().startswith("#")])
    # Remove // single-line comments
    no_preproc = re.sub(r'//.*', '', no_preproc)
    # Remove /* multi-line */ comments
    no_preproc = re.sub(r'/\*.*?\*/', '', no_preproc, flags=re.DOTALL)
    return no_preproc
```

1. **Parsing and CFG Construction:** The script first reads a target C file, cleans it by removing comments, and identifies "leaders" to partition the code into a sequence of basic blocks. It then constructs a Control Flow Graph by mapping the successor-predecessor relationships between these blocks.

```python
# ---------- CFG ----------
class Block:
    def __init__(self, bid): self.id, self.lines = bid, []
class CFG:
    def __init__(self):
        self.blocks, self.edges, self._next = {}, set(), 0
    def new_block(self):
        bid = f"B{self._next}"; self._next += 1
        self.blocks[bid] = Block(bid); return bid
    def add_edge(self, u, v): self.edges.add((u, v))

gen = CGenerator()

def stmt_text(node):
    try:
        txt = gen.visit(node)
        if not txt.endswith("}") and not txt.endswith(";"):
            txt += ";"
        return txt
    except Exception:
        return str(type(node).__name__) + ";"
```

```python
def build_seq(cfg, stmts):
    entry = last = None
    cur = cfg.new_block()
    def flush():
        nonlocal entry, last, cur
        if entry is None: entry = cur
        last = cur
        cur = cfg.new_block()
        cfg.add_edge(last, cur)
        last = cur
    for s in stmts or []:
        if isinstance(s, (c_ast.If, c_ast.While, c_ast.For, c_ast.DoWhile, c_ast.Switch)):
            if cfg.blocks[cur].lines:
                if entry is None: entry = cur
                last = cur
            else:
                if entry is None: entry = cur
                last = cur
            if isinstance(s, c_ast.If):
                e, l = build_if(cfg, s)
            elif isinstance(s, c_ast.While):
                e, l = build_while(cfg, s)
            elif isinstance(s, c_ast.For):
                e, l = build_for(cfg, s)
            elif isinstance(s, c_ast.DoWhile):
                e, l = build_dowhile(cfg, s)
            else:
                e, l = build_switch_flat(cfg, s)
            cfg.add_edge(last, e)
            cur = l
            last = cur
        else:
            cfg.blocks[cur].lines.append(stmt_text(s))
    if entry is None:
        entry = cur
    return entry, cur
```

```python
def build_if(cfg, node: c_ast.If):
    cond = cfg.new_block(); cfg.blocks[cond].lines.append(f"if({gen.visit(node.cond)})")
    t_entry, t_exit = build_seq(cfg, _as_list(node.iftrue))
    cfg.add_edge(cond, t_entry)
    if node.iffalse is not None:
        f_entry, f_exit = build_seq(cfg, _as_list(node.iffalse))
        cfg.add_edge(cond, f_entry)
        join = cfg.new_block()
        cfg.add_edge(t_exit, join); cfg.add_edge(f_exit, join)
        return cond, join
    else:
        join = cfg.new_block()
        cfg.add_edge(cond, join)
        cfg.add_edge(t_exit, join)
        return cond, join

def build_while(cfg, node: c_ast.While):
    cond = cfg.new_block(); cfg.blocks[cond].lines.append(f"while({gen.visit(node.cond)})")
    b_entry, b_exit = build_seq(cfg, _as_list(node.stmt))
    cfg.add_edge(cond, b_entry)
    cfg.add_edge(b_exit, cond)
    exitb = cfg.new_block()
    cfg.add_edge(cond, exitb)
    return cond, exitb

def build_for(cfg, node: c_ast.For):
    initb = cfg.new_block()
    if node.init is not None:
        for s in _as_list(node.init): cfg.blocks[initb].lines.append(stmt_text(s))
    condb = cfg.new_block(); cfg.blocks[condb].lines.append(f"for({gen.visit(node.cond) if node.cond else ''})")
    cfg.add_edge(initb, condb)
    b_entry, b_exit = build_seq(cfg, _as_list(node.stmt))
    cfg.add_edge(condb, b_entry)
    postb = cfg.new_block()
    if node.next is not None:
        for e in _as_list(node.next): cfg.blocks[postb].lines.append(stmt_text(s))
    cfg.add_edge(b_exit, postb)
    cfg.add_edge(postb, condb)
    exitb = cfg.new_block(); cfg.add_edge(condb, exitb)
    return initb, exitb
```

```python
def build_dowhile(cfg, node: c_ast.DoWhile):
    body_entry, body_exit = build_seq(cfg, _as_list(node.stmt))
    condb = cfg.new_block(); cfg.blocks[condb].lines.append(f"do_while({gen.visit(node.cond)})")
    cfg.add_edge(body_exit, condb)
    cfg.add_edge(condb, body_entry)
    exitb = cfg.new_block(); cfg.add_edge(condb, exitb)
    return body_entry, exitb

def build_switch_flat(cfg, node: c_ast.Switch):
    h = cfg.new_block(); cfg.blocks[h].lines.append(f"switch({gen.visit(node.cond)})")
    body_entry, body_exit = build_seq(cfg, _as_list(node.stmt))
    cfg.add_edge(h, body_entry)
    exitb = cfg.new_block(); cfg.add_edge(body_exit, exitb)
    return h, exitb

def _as_list(x):
    if x is None: return []
    if isinstance(x, c_ast.Compound): return x.block_items or []
    return [x]

def build_cfg_from_source(src):
    parser = c_parser.CParser()
    cleaned = strip_preprocessor(src)
    ast = parser.parse(cleaned)
    cfg = CFG()
    funcs = [d for d in ast.ext if isinstance(d, c_ast.FuncDef)]
    mains = [f for f in funcs if getattr(getattr(f.decl, 'name', None), 'lower', lambda: '')() == 'main' or f.decl.name == 'main']
    targets = mains if mains else funcs
    entry = last = None
    for f in targets:
        e, l = build_seq(cfg, _as_list(f.body))
        if entry is None: entry = e
        if last is not None: cfg.add_edge(last, e)
        last = l
    if entry is None:
        entry = cfg.new_block()
    return cfg
```

2. **CFG Visualization:** The `graphviz` library is used to generate a `.dot` representation of the CFG, which is then rendered as an image file for visualization.

```python
# ---------- DOT ----------
def to_dot(cfg):
    out = ["digraph CFG {", '  node [shape=box, fontname="Consolas", fontsize=10];', "  rankdir=TB;"]
    for b, blk in cfg.blocks.items():
        escaped_lines = []
        for ln in blk.lines:
            safe = ln.replace('\\', '\\\\').replace('"', '\\"')
            escaped_lines.append(safe)
        lbl = "\\l".join(escaped_lines) + ("\\l" if escaped_lines else "")
        if not lbl:
            lbl = "(join/ε)"
        out.append(f'  "{b}" [label="{b}: {lbl}"];')
    for u, v in sorted(cfg.edges):
        out.append(f'  "{u}" -> "{v}";')
    out.append("}")
    return "\n".join(out)

def render_dot(dot_text, png_path):
    dot_path = png_path[:-4] + ".dot"
    write_text(dot_path, dot_text)
    try:
        subprocess.run(["dot", "-Tpng", dot_path, "-o", png_path], check=True)
        print(f"[SUCCESS] Rendered {png_path}")
    except Exception as e:
        print(f"[WARN] Graphviz render failed: {e}")
```

3. **Cyclomatic Complexity Calculation:** The script automatically counts the number of nodes (N) and edges (E) in the generated CFG and computes the Cyclomatic Complexity using the formula $CC = E - N + 2$.

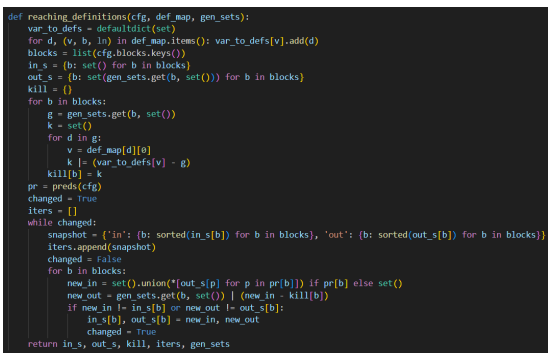
```python
# ---------- Metrics ----------
def compute_metrics(cfg):
    N, E = len(cfg.blocks), len(cfg.edges)
    CC = E - N + 2
    return {"N": N, "E": E, "CC": CC}
```

4. **Reaching Definitions Analysis:**

   - **Definition Identification:** The tool scans all basic blocks to identify every variable assignment and assigns each a unique definition ID (e.g., 'D0', 'D1').
   - **Gen/Kill Set Computation:** For each basic block B, it computes the `gen[B]` set (definitions created within B) and the `kill[B]` set (definitions of the same variables that exist elsewhere).
   - **Iterative Dataflow Analysis:** The core of the analyzer is an iterative algorithm that applies the following dataflow equations until the 'in' and 'out' sets for all blocks stabilize (reach a fixed point):

$$\text{in}[B] = \bigcup_{P \in \text{pred}(B)} \text{out}[P]$$

$$\text{out}[B] = \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B])$$



2.4 Execution

The analyzer was executed from the terminal for each of the three C programs. The script successfully generated all required artifacts for each program, including the CFG image, definition mappings, and CSV files containing the final analysis data.

```
(.venv) PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab7> python tool\cs202_lab7.py corpus\program1.c corpus\program2.c cor
pus\program3.c

[PROCESSING] program1
[SUCCESS] Rendered C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program1\program1_cfg.png
[METRICS] N=141 E=170 CC=31
[OUTPUTS] All files written to C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program1/

[PROCESSING] program2
[SUCCESS] Rendered C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program2\program2_cfg.png
[METRICS] N=65 E=80 CC=17
[OUTPUTS] All files written to C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program2/

[PROCESSING] program3
[SUCCESS] Rendered C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program3\program3_cfg.png
[METRICS] N=101 E=126 CC=27
[OUTPUTS] All files written to C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out\program3/

[SUMMARY] Wrote consolidated metrics to C:\Users\mdsib\OneDrive\Desktop\cs202_lab7\out/metrics_summary.csv

[DONE] All programs processed. Check out/ for organized folders.
```

3 Results

3.1 CFG and Cyclomatic Complexity

The analyzer successfully generated CFGs for all three programs and calculated their Cyclomatic Complexity. The summary of these metrics is presented below

Program No.	Program Name	No. of Nodes (N)	No. of Edges (E)	Cyclomatic Complexity (CC)
1	program1	141	170	31
2	program2	65	80	17
3	program3	101	126	27

Figure 1: Cyclomatic Complexity Metrics Summary

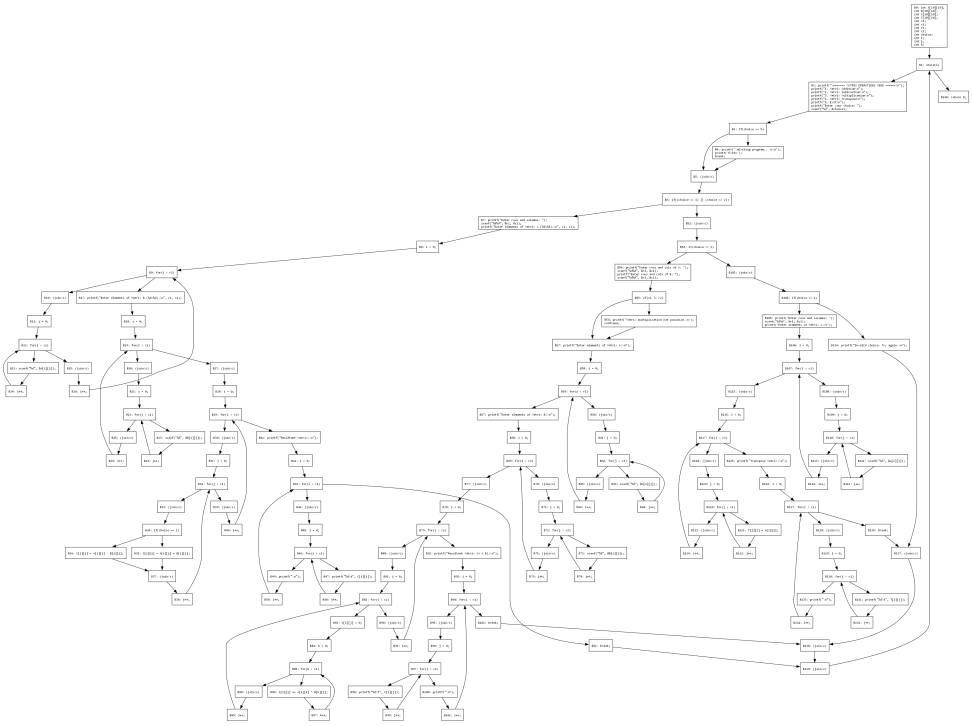


Figure 2: Control Flow Graph for `program1.c`

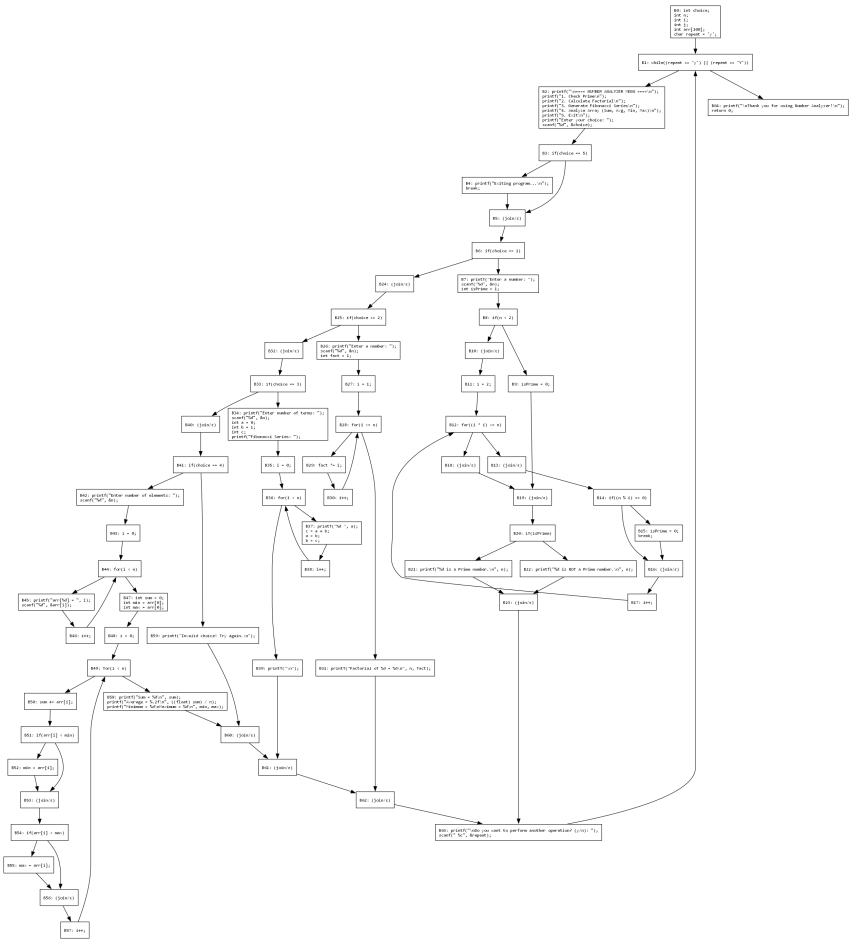


Figure 3: Control Flow Graph for `program2.c`

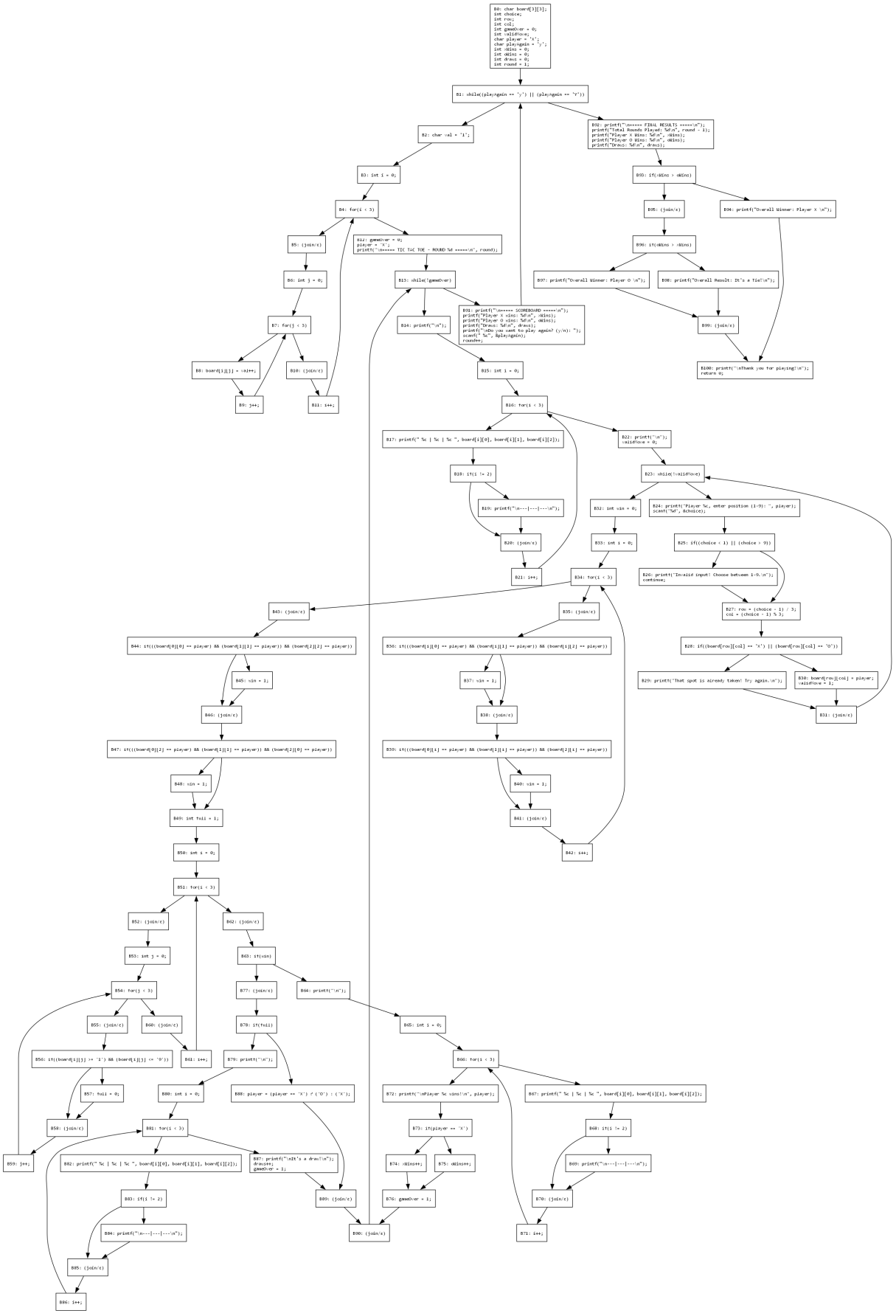


Figure 4: Control Flow Graph for `program3.c`

3.2 Reaching Definitions Analysis

The script successfully performed Reaching Definitions Analysis for all programs. As an illustrative example, the results for `program2.c` (Number Analyzer) are highlighted below. First, variable definitions were mapped to unique IDs

```
D0: var=i, block=B8, line=i = 0;
D1: var=j, block=B11, line=j = 0;
D2: var=i, block=B18, line=i = 0;
D3: var=j, block=B21, line=j = 0;
D4: var=i, block=B28, line=i = 0;
D5: var=j, block=B31, line=j = 0;
D6: var=i, block=B42, line=i = 0;
D7: var=j, block=B45, line=j = 0;
D8: var=i, block=B58, line=i = 0;
D9: var=j, block=B61, line=j = 0;
D10: var=i, block=B68, line=i = 0;
D11: var=j, block=B71, line=j = 0;
D12: var=i, block=B78, line=i = 0;
D13: var=j, block=B81, line=j = 0;
D14: var=k, block=B84, line=k = 0;
D15: var=i, block=B93, line=i = 0;
D16: var=j, block=B96, line=j = 0;
D17: var=i, block=B106, line=i = 0;
D18: var=j, block=B109, line=j = 0;
D19: var=i, block=B116, line=i = 0;
D20: var=j, block=B119, line=j = 0;
D21: var=i, block=B126, line=i = 0;
D22: var=j, block=B129, line=j = 0;
```

Figure 5: Definition IDs for `program1.c`

```
D0: var=repeat, block=B0, line=char repeat = 'y';
D1: var=isPrime, block=B7, line=int isPrime = 1;
D2: var=isPrime, block=B9, line=isPrime = 0;
D3: var=i, block=B11, line=i = 2;
D4: var=isPrime, block=B15, line=isPrime = 0;
D5: var=fact, block=B26, line=int fact = 1;
D6: var=i, block=B27, line=i = 1;
D7: var=a, block=B34, line=int a = 0;
D8: var=b, block=B34, line=int b = 1;
D9: var=i, block=B35, line=i = 0;
D10: var=c, block=B37, line=c = a + b;
D11: var=a, block=B37, line=a = b;
D12: var=b, block=B37, line=b = c;
D13: var=i, block=B43, line=i = 0;
D14: var=sum, block=B47, line=int sum = 0;
D15: var=min, block=B47, line=int min = arr[0];
D16: var=max, block=B47, line=int max = arr[0];
D17: var=i, block=B48, line=i = 0;
D18: var=min, block=B52, line=min = arr[i];
D19: var=max, block=B55, line=max = arr[i];
```

Figure 6: Definition IDs for `program2.c`

```
D0: var=gameOver, block=B0, line=int gameOver = 0;
D1: var=player, block=B0, line=char player = 'X';
D2: var=playAgain, block=B0, line=char playAgain = 'y';
D3: var=xWins, block=B0, line=int xWins = 0;
D4: var=oWins, block=B0, line=int oWins = 0;
D5: var=draws, block=B0, line=int draws = 0;
D6: var=round, block=B0, line=int round = 1;
D7: var=val, block=B2, line=char val = '1';
D8: var=i, block=B3, line=int i = 0;
D9: var=j, block=B6, line=int j = 0;
D10: var=gameOver, block=B12, line=gameOver = 0;
D11: var=player, block=B12, line=player = 'X';
D12: var=i, block=B15, line=int i = 0;
D13: var=validMove, block=B22, line=validMove = 0;
D14: var=row, block=B27, line=row = (choice - 1) / 3;
D15: var=col, block=B27, line=col = (choice - 1) % 3;
D16: var=validMove, block=B30, line=validMove = 1;
D17: var=win, block=B32, line=int win = 0;
D18: var=i, block=B33, line=int i = 0;
D19: var=win, block=B37, line=win = 1;
D20: var=win, block=B40, line=win = 1;
D21: var=win, block=B45, line=win = 1;
D22: var=win, block=B48, line=win = 1;
D23: var=full, block=B49, line=int full = 1;
D24: var=i, block=B50, line=int i = 0;
D25: var=j, block=B53, line=int j = 0;
D26: var=full, block=B57, line=full = 0;
D27: var=i, block=B65, line=int i = 0;
D28: var=gameOver, block=B76, line=gameOver = 1;
D29: var=i, block=B80, line=int i = 0;
D30: var=gameOver, block=B87, line=gameOver = 1;
D31: var=player, block=B88, line=player = (player == 'X') ? ('O') : ('X');
```

Figure 7: Definition IDs for `program3.c`

After running the iterative analysis until convergence, the final ‘in‘ and ‘out‘ sets were computed. Below shows a selection of these results for key basic blocks.

Basic-Block	gen[B]	kill[B]	in[B]	out[B]
B0				
B1			D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9	D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9
B10			D0 D1 D11 D13 D14 D16 D18 D20 D22 D3 D5 D7 D9	D0 D1 D11 D13 D14 D16 D18 D20 D22 D3 D5 D7 D9
B100			D14 D15 D16	D14 D15 D16
B101			D14 D15 D16	D14 D15 D16
B102			D1 D11 D13 D14 D15 D16 D18 D20 D22 D3 D5 D7 D9	D1 D11 D13 D14 D15 D16 D18 D20 D22 D3 D5 D7 D9
B103			D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9	D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9
B104			D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9	D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9
B105			D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9	D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9
B106	D17	D0 D10 D12 D15 D19 D2 D21 D4 D6 D8	D1 D11 D13 D14 D15 D16 D18 D20 D21 D22 D3 D5 D6 D7 D9	D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9
B107			D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9	D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9
B108			D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9	D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9
B109	D18	D1 D11 D13 D16 D20 D22 D3 D5 D7 D9	D1 D11 D13 D14 D16 D17 D18 D20 D22 D3 D5 D7 D9	D14 D17 D18

Figure 8: Sample of Final Reaching Definitions for `program1.c`

Basic-Block	gen[B]	kill[B]	in[B]	out[B]
B0	D0			D0
B1			D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D2 D3 D4 D5 D6 D7 D8 D9	D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D2 D3 D4 D5 D6 D7 D8 D9
B10			D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D3 D5 D6 D7 D8 D9	D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D3 D5 D6 D7 D8 D9
B11	D3	D13 D17 D6 D9	D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D3 D5 D6 D7 D8 D9	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D5 D7 D8
B12			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B13			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B14			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B15	D4	D1 D2	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B16			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B17			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B18			D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8	D0 D1 D10 D11 D12 D14 D15 D16 D18 D19 D3 D4 D5 D7 D8
B19			D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D2 D3 D4 D5 D6 D7 D8 D9	D0 D1 D10 D11 D12 D14 D15 D16 D17 D18 D19 D2 D3 D4 D5 D6 D7 D8 D9

Figure 9: Sample of Final Reaching Definitions for `program2.c`

Basic-Block	gen[B]	kill[B]	in[B]	out[B]
B0	D0 D1 D2 D3 D4 D5 D6	D10 D11 D28 D30 D31		D0 D1 D2 D3 D4 D5 D6
B1			D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9
B10			D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D26 D28 D3 D30 D31 D4 D5 D6 D7 D8 D9	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D26 D28 D3 D30 D31 D4 D5 D6 D7 D8 D9
B100			D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9
B11			D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D26 D28 D3 D30 D31 D4 D5 D6 D7 D8 D9	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D26 D28 D3 D30 D31 D4 D5 D6 D7 D8 D9
B12	D10 D11	D0 D1 D28 D30 D31	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D8 D9	D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D3 D4 D5 D6 D7 D8 D9
B13			D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9	D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9
B14			D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9	D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9
B15	D12	D18 D24 D27 D29 D8	D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B16			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B17			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B18			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B19			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B2	D7		D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8	D0 D1 D10 D11 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D24 D25 D26 D27 D28 D29 D3 D30 D31 D4 D5 D6 D7 D8 D9
B20			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B21			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B22	D13	D16	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B23			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B24			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B25			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B26			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B27	D14 D15		D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B28			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9
B29			D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9	D10 D11 D12 D13 D14 D15 D16 D17 D19 D2 D20 D21 D22 D23 D25 D26 D28 D3 D30 D31 D4 D5 D6 D7 D9

Figure 10: Sample of Final Reaching Definitions for `program3.c`

4 Interpretation of Results

The Cyclomatic Complexity values in Figure 1 align with the intuitive complexity of the programs. `program1.c` **(Matrix Operations)** has the highest CC of 31, which is expected due to its numerous nested loops for matrix traversal and the multiple branches in its main menu. `program3.c` **(Tic-Tac-Toe)** also has a high CC of 27, reflecting its complex game logic for checking win/draw conditions and managing player turns.

The Reaching Definitions analysis provides precise insights into the data flow. For instance, in `program2.c`, consider the loop responsible for checking if a number is prime (starting around block B11). The ‘in‘ set for the loop header block (B11) contains two definitions for the loop counter ‘i‘: one from before the loop begins and another from the increment at the end of the loop body. This correctly shows that the loop can be reached from two different paths, and the value of ‘i‘ at the start of an iteration could be its initial value or a value from a previous iteration.

This analysis is crucial for identifying potential software bugs. If a variable is used within a block, the ‘in‘ set for that block tells us exactly which prior assignments could have produced its value. If the ‘in‘ set contains an unexpected or uninitialized definition, it points to a potential use-before-define error or a logical flaw in the program’s control flow.

5 Conclusion

This lab successfully achieved its objective of creating a functional program analysis tool. A Python script was developed to automatically construct CFGs, calculate Cyclomatic Complexity, and perform a complete Reaching Definitions Analysis on a given C program. The tool was validated by running it on three non-trivial C programs, and it produced accurate and insightful results. The process demonstrated the practical application of dataflow analysis in understanding how variable values propagate through complex control structures, laying the foundation for more advanced static analysis techniques like bug detection and code optimization.