

Live Variable Analysis UI

Raadhes Chandaluru
CS22B069

IIT Madras

02/11/2025

Creating Educational UI for CFG and Live Variable Analysis

- Frontend is in Python Dash which is an interactive web application framework
- C Code is input by user in frontend
 - Python uses clang on the backend to lower C code to LLVM IR
 - This is passed through our live variable LLVM Pass
- LLVM Pass to generate information per iteration
 - Changes in information populated into graphviz dot file format
- Python graphviz renders .dot files
- Provided a slot to give clang flags in case higher optimization levels want to be seen in CFG (Or other flags)

AI assistance in UI code

Challenges & Final Setup

LLVM C Code CFG & Live Variable Per Iteration

Show / Hide Instructions

Enter C Code:

```
#include <stdio.h>
int main() {
    int x = 3;
    if(x > 0){
        x = x + 1;
    }
    return x;
}
```

Clang Compile Flags: -O0

Generate CFG

{ Prev cfg_main0.dot Next }

The screenshot shows the LLVM C Code CFG & Live Variable Per Iteration tool interface. At the top, there's a code editor with the provided C code and a clang compile flags input field set to '-O0'. Below the code editor are 'Show / Hide Instructions' and 'Generate CFG' buttons. Underneath, there's a navigation bar with 'Prev' and 'Next' buttons and a dropdown menu currently showing 'cfg_main0.dot'. The main area displays three nodes of a Control Flow Graph (CFG) as rounded rectangles with a pink background. The first node contains assembly code for allocation and initialization of variables %1 and %2. The second node contains assembly code for addition and store operations involving %2 and %7. The third node contains a single store operation involving %9. Arrows indicate the flow from the first node to the second, and from the second to the third.

- Changes such as `DbgVariableRecord` not present in LLVM-14, but part of LLVM-20. Upgraded to LLVM-20 by installing from [llvm.sh](#)
- Generation of CFG manually slightly painful but not too hard