Project 3
Advanced Compilers

Jun 4, 2025
Due: Jun 21, 2025

In previous project, we had found integer constants throughout the LLVM IR using Sparse Conditional Constants Propagation (SCCP). In this project, we will fold (propagate) constants with '`SimpleSCCP`' analysis result. Thanks to SSA formed LLVM IR and LLVM's abundant APIs, manipulating variables is quite simple.

1 LLVM Pass/Analysis Manager

LLVM's middle-end, responsible for machine-independent optimizations, has two instances of `PassManager`[1]; `FunctionPassManager` and `ModulePassManager`. Those two pass managers are reponsible for function and module, respectively. Similarly, there are `FunctionAnalysisManager` and `ModuleAnalysisManager`, reponsible for the analyses over their IR units.

`PassManager` manages registered passes and calls them when they are needed as requested in optimization pipeline. While it is discouraged to call another pass inside one pass (i.e., nesting passes), a pass can freely request and use analyses results via `AnalysisManager`. However, a pass' modification (transformation) on IR can make the results of analyses incorrect. Therefore, a pass should invalidate corresponding analysis via analysis manager, if the pass had affetcted the result of other analyses.

For your information, LLVM had changed its pass (analysis) manager. Through these projects, we used and will use the new pass manager only.

2 SSA Form and Value Substitution

In SSA form, diffent values of the same variable are statically discriminated by their own name.

$$\boxed{\begin{array}{l} a = 4 \\ x = a + 2 \\ a = 7 \\ y = a + 2 \end{array}} \qquad \boxed{\begin{array}{l} a_0 = 4 \\ x_0 = a_0 + 2 \\ a_1 = 7 \\ y_0 = a_1 + 2 \end{array}}$$

(a) Three-address code form (b) SSA form

Figure 1: Example IR snippet

(a) is a code snippet in traditional normal (three-address code; TAC) form, and (b) is the same code but in SSA form. From the code, the variable x and y are being assigned the same expression $a + 2$. However, the value of x and y are 6 and 9, respectively. Although, x and y have different values, one cannot easily discriminate those values easily. However, in SSA form, those values are easily distinguishable, since their expressions are different. The key difference here is that a_0 and

[1]There are more template instances of `PassManager`, used in back-end. (`llc`)

a_1 had their own name, unlike in (a). Therefore, when we substitute 'a's with their value (constant folding), we can easily decide the group of expressions to be substituted.

3 Code

3.1 `SimpleSCCPTransform`

Class for the costant propagation transformation pass.

3.2 `SimpleSCCPTransform::run`

Pass' main function. If the pass modified any of the IR, the pass should invalidate the previous analyses by returning '`PreservedAnalyses::none`'.

3.3 `SimpleSCCPTransform::foldConstants`

Core function which replaces *Use*s of the variables with known constants. One can easily remove the instruction from the basic block via '`removeFromParent`'. While substitution, one may need to replace branch (`br`) instructions. In LLVM, this instruction replacement or creation can be done through '`IRBuilder`'.

Tips

- Basic block should end with the 'terminator' instruction.
- Basic blocks with zero reference count will be dropped automatically.
- Removing an iterator (or the corresponding element) while iterating over its range can invalidate the iterator.
- One can find LLVM's implementation at 'lib/Transforms/Scalar/SCCP.cpp'.

4 Input

As same as project2, your pass will get an IR, which passed '`mem2reg`'.

5 Objectives

- Copy and paste your `visitPHINode`, `analyze`, `visit` from the project 2.
- Implement a `TODO` at `foldConstants` in `SimpleSCCP.cpp`.
  - Your result should be the same as LLVM's `sccp` result.
  - Transformed IR should be able to be compiled and executed normally.

6 Commands

- Build your plugin.
  ```
  cmake -S project3 -G Ninja -B build
  ```
- Generate IR from the source.
  ```
  clang -O0 -S -emit-llvm -Xclang -disable-O0-optnone -Xclang -disable-llvm-passes
  -fno-discard-value-names test.c -o test.ll
  ```

- Run 'mem2reg' pass.
  `opt -S -passes='mem2reg'./test.ll -o input.ll`
- Call your pass from `opt`.
  `opt -S -load-pass-plugin build/lib/libSimpleSCCP.so -passes='simple-sccp' ./input.ll -o out.ll`
- Run '`verify`'.
  `opt -S -passes='verify' out.ll`
- Generate IR with LLVM's `sccp`.
  `opt -S -passes=sccp ./input.ll -o llvm_out.ll`
- Compile the output IR.
  `clang ./out.ll -o test`

7 Submission

- Compress and submit your `SimpleSCCP.cpp` as `PR3_<student_id>.zip` (e.g., `PR3_2024-12345.zip`) at eTL.
- If you have modified any file other than `SimpleSCCP.cpp` or added new files, compress and submit your project directory, including **CMakeLists.txt**, into a single zip file named after your student id (e.g., `PR3_2024-23456.zip`).

8 Example

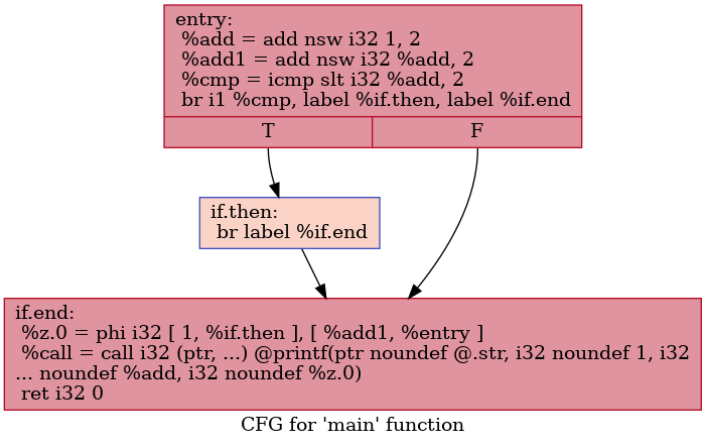

Figure 2: CFG of `input.ll`

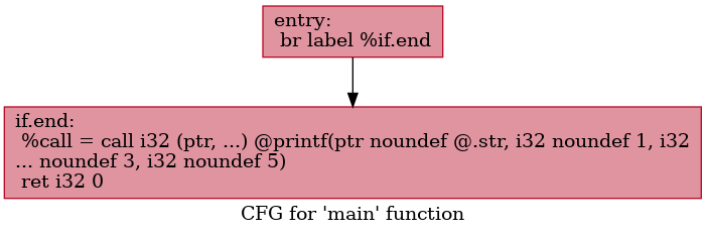

Figure 3: CFG of `out.ll`