Assignment 1: Introduction to Basics of LLVM

Introduction to Program Analysis and Optimization

January 13, 2024

Abstract

LLVM (Low Level Virtual Machine) is an open-source compiler infrastructure that provides a set of tools and libraries for building compilers, debuggers, and other software development tools. Initially conceived at the University of Illinois at Urbana, LLVM comprises a collection of adaptable and reusable technologies for constructing diverse compilers and related software utilities.

We will heavily rely on the LLVM Compiler infrastructure for our programming assignments. Ideally, you should have access to an x86-based machine, preferably operating on Linux. The primary goal of this initial assignment is to acquaint you with the LLVM infrastructure. The assignment involves installing LLVM from the ground up and exploring its analysis and optimization techniques. This exercise serves as a practical introduction to LLVM's capabilities and functionalities.

1 LLVM Framework Installation (0 pts)

1.1 Required Softwares

We suggest you to check the software requirements for LLVM installation. You can use a conda environment to install the prerequisite packages.

```
conda create -n pa python=3.10
conda activate pa
pip install cmake
```

1.2 Cloning Into Repositories

Clone the LLVM repository from Git Hub and change the version to release/14.x. We need to change llvm-version to release/14.x in order to ensure consistency among all students. To clone the LLVM repository and change the version, run the two commands below.

```
git clone https://github.com/llvm/llvm-project.git -b release/14.x
```

The repository is around 1.27 GB in size. Before cloning, make sure you have good network connectivity.

1.3 Building LLVM

After successfully cloning LLVM-14, the next step is to build both LLVM and clang from the ground up. LLVM employs CMake to create a build system, allowing you to specify the desired build system for compiling the LLVM source files. We recommend using "Unix Makefiles" as the build system for Ubuntu. To build LLVM and clang, execute the following command:

```
cmake -S llvm -B build -G "Unix Makefiles" -DLLVM_ENABLE_PROJECTS="clang"
    -DLLVM_TARGETS_TO_BUILD="X86" -DLLVM_CCACHE_BUILD=ON
cd build
```

```
make -j4
```

`cmake` provides multiple options to build the files. One such option is `-DLLVM_ENABLE_PROJECTS="..."`. You can specify additional LLVM subprojects for building by providing a semicolon-separated list, including any of: clang, clang-tools-extra, lldb, lld, polly, or cross-project-tests. We suggest building only `clang` for now.

After you have successfully executed `cmake`, you need to enter `build` folder and use `make -jn` command to build LLVM and clang. The -j flag used in the make command is used to specify the number of parallel jobs (threads) that make can initiate simultaneously when building a project. For instance, if you run make -j4, it instructs make to start up to 4 concurrent jobs (threads) to build the target(s) specified in the makefile. The number after the -j flag indicates the maximum number of jobs that can run simultaneously. If you omit a number after -j, `make` will execute as many jobs as the number of processor cores available in your system by default. To get more information about LLVM installation, you can visit the following link.

2 Generating & Analyzing LLVM IR (80 pts)

Your initial assignment involves creating and examining LLVM Intermediate Representation (IR) for input C/C++ files. We provide five C/C++ files for this purpose. Your objective is to compile these C/C++ files utilizing LLVM clang, resulting in LLVM IR files with a `.ll` extension. It is essential to refrain from generating files with the `.bc` extension and avoid relying on a prebuilt version of clang. You can compile C and C++ files using the following command with clang:

```
../build/bin/clang -S -emit-llvm file_name.c -o file_name.ll
../build/bin/clang++ -S -emit-llvm file_name.cpp -o file_name.ll -I/usr/include/c++/ver_number
    -I/usr/include/x86_64-linux-gnu/c++/ver_number
```

Please replace the 'ver_number' in the command above with the CPP version of your host machine. We have provided you with the partially filled commands of the same in the `run.sh` file. Your job is to fill `run.sh` file with suitable commands. After completing the above steps, you are required to respond to the following questions:

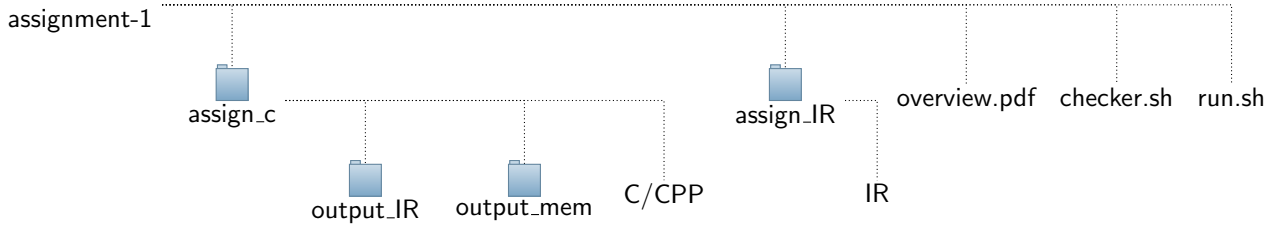
- You must have seen different types of instructions in LLVM IR, e.g., `alloca`, `load`, `store`, and `icmp` instructions. Mention the name and purpose of any five such instructions. (20 pts)
- Notice that function attributes are mentioned before defining each function. e.g, `; Function Attrs: noinline nounwind uwtable`. Name any two of such attributes and their purposes. (10 pts)
- Run the `-mem2reg` pass on the IR files and store transformed IR file in another `.ll` file. Write down the differences between these two files and clearly state the reason for such transformations. Hint: You need to run `-mem2reg` pass using `opt` tool. If you are finding that your transformed IR and original IR have no difference, then your `-mem2reg` pass could not probably transform the IR. You need to find out how to solve this problem on your own and complete the `run.sh` file such that the command executes successfully. You also need to mention the reason for such behaviour in `overview.pdf` file. (15+5 = 20 pts)
- List out the differences between an IR file of a C file and an IR file of a CPP file. (5 pts)
- You may have noticed that some function names in the IR file are depicted differently from those in the source file (Notice file3.cpp and the corresponding IR file). This phenomenon is called Name Mangling. Can you state the reason for this? Also, Can you state how name mangling is done? You need not provide an exact algorithm but should give an approximate idea of the same. (10 + 15 = 25 pts)

3 Guess the Functionality (20 pts)

You have already generated and analyzed IR codes for each given C/C++ code. To work on this part of the assignment, we will provide you with a different set of five Intermediate Representation (IR) codes. Your task is to analyze these IR representations and determine the code's functionality or operation. The IR files can be found inside the '`assign_IR`' folder, and you have to write the functionality of each IR file in your `overview.pdf` file. For example, if the IR file does an addition operation with two integers, you should write "addition of two integers". Note: Do **not** write paragraphs or the detail of each line of the IR code.

4 Submission Guidelines

The structure of the given template directory is mentioned above.



There are five C/CPP codes inside the `assign_c` folder, which you must use for the tasks mentioned in Section 2. Generated IR files should be kept inside the `output_IR` folder, and corresponding transformed IR files (after running `-mem2reg`) should be kept inside `output_mem` folder. Your generated IR files should have the same name as the corresponding C/CPP files (e.g., the IR file corresponding to file1.c should be named file1.ll). Similarly, transformed IR files should have the suffix "_mem" after the original C/CPP file name (e.g., the transformed IR file corresponding to file1.c should be named file1_mem.ll).

LLVM IR files in the `assign_IR` folder should be used for the tasks mentioned in Section 3. Please write the answers in the `overview.pdf` file.

Here are some DOs and DONTs for the assignment.

DOs

- Before uploading the assignment change the folder name to your ROLL_NO. Use git commit to upload the assignment.
- Run the script in `checker.sh` file and submit the assignment only after receiving an "Accept" output from the script. It checks the naming conventions and folder structure. Note: `checker.sh` cannot validate the correctness of your assignment results.
- Clone the assignment repository inside the llvm-project folder.
- TAs should only run `run.sh` file to generate IR files and transformed IR files. If the files are not automatically generated after running `run.sh`, your assignment may get zero marks.
- Write all the answers in the `overview.pdf` file.

DONTs

- Do not submit assignment if the `checker.sh` gives "Rejected". Your assignment will not be evaluated if your directory structure or naming conventions do not match.
- Do not change the name of any files or folders.
- Do not use the GPT tool to write the theory answers. There will be a heavy penalty for such behavior.