《高级程序设计 II》—大实验

第 1 次作业

2020 年 04 月 13 日

- 本次作业主要目的是让同学们通过编写代码，理解、掌握 LLVM 编程的基本方式、函数与基本块及指令的遍历，熟悉中间表示形式 LLVM IR 的基本含义
- 提交的实验报告为 PDF 文件，文件名格式为“学号-姓名-大实验-作业 1.pdf“，文件内容格式不限，但应该清晰标注学号、姓名与高级程序设计 II 大实验作业 1 等信息
- 作业截止时间 2020 年 04 月 22 日晚 23:59（周三），在没有明确通知可缓交的情况下，每晚交一天，扣 3 分；缓交扣分最多 30 分（即本次作业若拖到期末才上交，最多可得 70 分）
- 提交至 OBE 系统“作业”板块的“二、大实验作业 1”
- 其他注意事项：
  - 使用 clang 生成 LLVM 中间表示的命令如：`clang -emit-llvm -S test.c`，获得文本文件 test.ll

使用说明：

1. 将目录中的 Project 和 Testcase 文件夹放置$HOME 目录下
2. 对于自己编译的 LLVM，需要将编译出来的 clang 所在的路径添加到环境变量 PATH 中

   `export PATH=$PATH:{PATH_TO_YOUR_CLANG_DIR}`

   对于使用我们提供的虚拟机，或者 LLVM 编译脚本编译的同学，无需执行这一步

3. 在 Project 目录的 MyPass.cpp 中完成此次作业的主体内容，并在 Project 目录下编译

```
cd $HOME/Project
make
```

4. 在 Testcase 目录下的 test1 文件夹中启用编译好的 MyPass.so 对目标文件 test1/TestMe.c 进行分析

```
cd $HOME/Testcase/test1
./analyze.sh
```

（要求有可执行权限，也可使用"sh ./analyze.sh"方式执行）

基本知识：

深度优先遍历：

参考：https://baike.baidu.com/item/%E6%B7%B1%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/5224976

基本算法（以下展示了最基本的深度优先搜索的算法，通常使用递归的方式实现，未考虑遇到已遍历过的结点应如何处理）：

```
Dfs(node) {
    Process(node);
    Foreach (succ in node.successors) {
        Dfs(succ);
    }
}
```

广度优先遍历：

参考：https://baike.baidu.com/item/%E5%AE%BD%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2/5224802

基本算法（以下展示了最基本的广度优先搜索的算法，通常使用 worklist 的方式实现，未考虑遇到已遍历过的结点应如何处理）：

```
Bfs(node) {
    Worklist wl;
    wl.insert(node);
    while (wl.not_empty()) {
```

```
        node = wl.pop_first();
        Process(node);
        Foreach (succ in node.successors) {
            wl.push_last(succ);
        }
    }
}
```

其他：

给定 BasicBlock，获取后继结点（successor）：

https://blog.csdn.net/wuhui_gdnt/article/details/77094940

获取前驱结点：https://www.jianshu.com/p/6399a4ec2e90（需要相关头文件。在 LLVM Manual 上也有相应的示例）

自学：

1. 调用 Instruction、BasicBlock、Function 等数据结构中的各个函数，了解其作用；
2. 请自学 LLVM 中另一种重要的数据结构 Value，并通过 Instruction 相关函数，了解并掌握 Value 及其与 Instruction 中各元素的关系与作用。

作业中附件说明：

附件给出了一个简单的 C 代码文件（Test.c）、对应的 LLVM IR 文件（Test.ll）以及根据 DFS 和 BFS 遍历每个具有函数体的函数的 BasicBlock 及其指令（Instruction）的结果（Test-result.txt）。结果中，对于每个 BasicBlock，首先表明了当前 BasickBlock 的地址（不同的运行会产生不一致的地址结果），然后分别获取当前 BasickBlock 的前驱结点与后继结点，接下来输出当前 BasicBlock 中每条 Instruction。如下所示：

```
+ Visit BB0x7fffc7b58e80
  # Predecessors:
    : 0x7fffc7b3a360
  # Successors:
    : 0x7fffc7b58ed0
  # Instructions:
    :   %15 = load i32, i32* %3, align 4
    :   %16 = call i32 @foo(i32 %15)
    :   store i32 %16, i32* %5, align 4
    :   br label %20
```

对于重复遇到的基本块，示例输出中如下：

```
+ Visit BB0x7fffc7b58ed0
  - It has been visited. Skip.
```

附件的 analyze.sh 脚本文件执行 clang 时使用了"-g"选项，可对比不使用该选项时输出结果的区别或是生成的 LLVM IR 的区别。注意，该脚本文件不能不经修改用于分析 Test.c 文件。

作业：

请根据相关资料，学习利用 LLVM 各种数据结构及方法，遍历 Module 中的 Function、Function 中的 BasicBlock、BasicBlock 中的 Instruction，使用深度优先搜索（DFS）或广度优先搜索(BFS)的方式遍历 Function 中的 BasicBlock，针对给定代码 **TestMe.c**，输出如示例结果所示的结果，并将代码和结果文件(截图或输出到文件均可)打包上传到 OBE 中。

注意：未实现 DFS 或 BFS，仅仅通过 Function 的迭代器（iterator）顺序遍历 BasicBlock 的（如下所示），最多获得本次作业总分的 70%（对于缓交者，在此基础上执行缓交扣分）。

```
for (auto iter = F.begin(); iter != F.end(); iter++) {
    BasicBlock &bb = *iter;
    Process(bb);
}
```