高级程序设计II 大实验作业四

项目地址：Github

本次作业分工：

农钧翔 2019201407：库函数对接与格式化输出

于倬浩 2019201409：图形化界面的实现 (见最终展示)

汪元森 2019201420：缺陷检测算法的实现

实验环境

操作系统：Ubuntu 20.04 LTS

```
$ clang --version
clang version 11.0.0 (https://github.com/llvm/llvm-project.git
ca376782ff8649d1a5405123f06a742e0e94b701)
Target: x86_64-unknown-linux-gnu
Thread model: posix
```

根据规则检测缺陷

负关联规则，即 $\{A\} \rightarrow \{B\}$ ，即需要检测有哪些代码块能够匹配项集$\{A, B\}$；正关联规则，即 $\{A\} \rightarrow \{B\}$，即需要检测有哪些代码块能匹配 $\{A\}$ 而不能匹配 $\{A, B\}$。

不难发现，检测部分需要实现的功能与计算支持度所用到的项集匹配功能大致框架相同。

算法设计

检测对负关联规则的违反，只需要对函数是否支持 A, B 分别进行判断即可。
检测对正关联规则的违反稍微复杂一些，这里给出一个例子：

```
x = A();
if( ... ) B(x);
else C(x);
```

如果存在正关联规则 $\{A\} \to \{B\}$，我们并不能说执行路径 $\{A\} \to \{C\}$ 违反了这个规则，因为执行 A 不必须执行 B。

基于上述讨论给出如下正关联规则检测算法：
如果一个函数支持 $\{A, B\}$，那么即使存在某条路径不支持 $\{A, B\}$ 而支持 $\{A\}$，我们也认为它是符合规则的。
如果某个函数不支持 $\{A, B\}$ 但支持 $\{A\}$，那么一定存在违反，对于这样的函数，我们将会指出某条执行路径上出现的错误。

实现细节

1. 在同一个函数中，如果**相同**的错误出现多次，我们只提供其中一个规则违反例，而不是全部规则违反例的组合。
   如果尽可能的列举出全部规则违反例的组合，将会导致同种错误重复出现多次，给使用者带来困扰。
2. 实现 `get_PARs_violations`, `get_NARs_violations` 函数，返回检测到的规则违反例 (即：违反规则的 instruction的集合)。
3. 对于函数 `retrieveDebugInfo`，即对于某个 instruction 返回 debug 信息的函数，用到了类 `llvm::MDNode` 与 `llvm::DISubprogram`，它们分别位于头文件 `llvm/IR/Metadata.h` 与 `llvm/IR/DebugInfo.h` 中。
4. 如果在被检测文件的某处出现了对缺少定义 (只有声明) 的函数的调用，输出结果会依旧显示行号，但将输出的文件名改为 *Linker Error*。
5. 使用 `outs()` 而不是 `errs()` 进行输出，避免和 clang 共享标准错误输出流。

运行结果

File: `Test6.out`

```
Error #1:
  %5 = call i8* @malloc(i32 %4), !dbg !18, TestMe.c:62
except an instruction but MISSING: free(i8*)

Error #2:
with the(se) following UNEXPECTED instruction(s):
  call void @free(i8* %12), !dbg !25, TestMe.c:93
  call void @free(i8* %9), !dbg !22, TestMe.c:92
```

File: `Test7.out`

```
Error #1:
  %6 = call i32 @is_valid(i32 %5), !dbg !19, TestMe.c:140
  call void @bar(i32 %12), !dbg !29, *Linker Error*:144
with the(se) following UNEXPECTED instruction(s):
  call void @foo(i32 %13), !dbg !31, *Linker Error*:145
```