

高级程序设计II 大实验-第一次作业

2019201409 于倬浩

前言

虽然这次作业，只要求上交代码和运行结果，但是老师在讲解说明时，我有很多具体的概念理解的并不深入，于是决定重新学习一次，并记录下学习的过程，包括中途涉及到的参考资料，以便之后参考。如果想直接看最终程序和运行结果，请跳转[这里](#)。所有涉及的文件均已上传至[这个GitHub仓库](#)。

实验环境

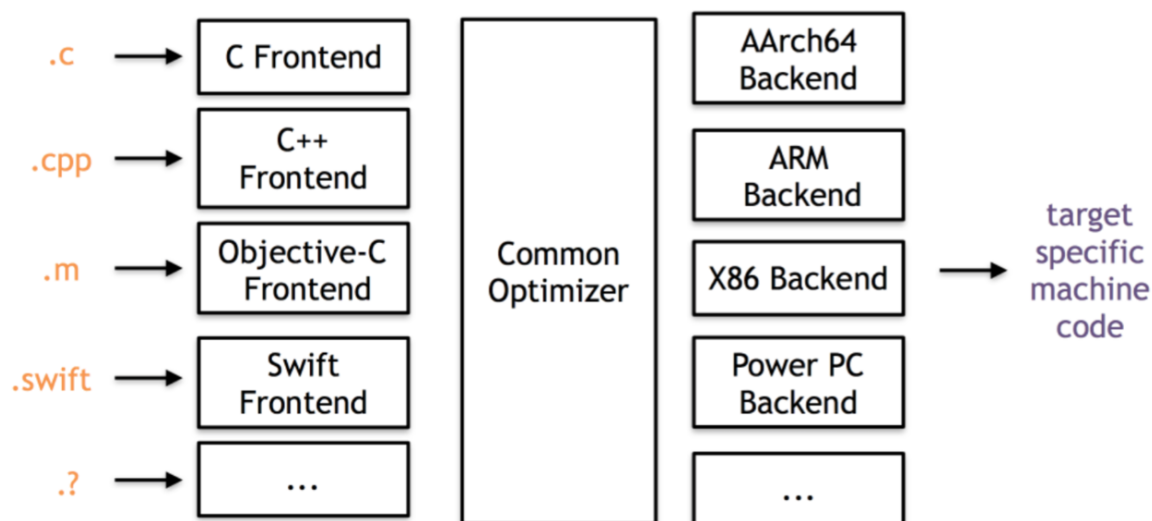
因为虚拟机的延迟比较难受，所以直接装了双系统。

```
1 me@Narcissus-OMEN-by-HP-Laptop:~$ cat /etc/issue
2 Ubuntu 19.10 \n \l
3
4 me@Narcissus-OMEN-by-HP-Laptop:~$ clang -v
5 clang version 11.0.0 (https://github.com/llvm/llvm-project.git
   ca376782ff8649d1a5405123f06a742e0e94b701)
6 Target: x86_64-unknown-linux-gnu
7 Thread model: posix
8 InstalledDir: /usr/local/bin
9 Found candidate GCC installation: /usr/lib/gcc/i686-linux-gnu/9
10 Found candidate GCC installation: /usr/lib/gcc/x86_64-linux-gnu/9
11 Selected GCC installation: /usr/lib/gcc/x86_64-linux-gnu/9
12 Candidate multilib: .;@m64
13 Selected multilib: .;@m64
```

准备工作

首先明白我们在做什么。

先来看编译器可以划分出来的几个部分：Frontend负责把源代码转化为数据结构，Optimizer负责优化冗余的代码逻辑，后端把数据结构转化为机器码。其中LLVM的特别之处在于，可以通过不同的Frontend处理不同的语言，以便重用后面的部分。



接着，了解IR的概念：IR是intermediate representation的简写，Wikipedia给的解释是"The data structure or code used internally by a compiler or virtual machine to represent source code"。那么现在就很明确了，Frontend就是在生成IR，Optimizer是优化IR，Backend是把IR转化为机器码，整个过程就是在不断交换、处理IR。

我们这次的大作业实际上应该是在写一个Pass。官方Docs已经解释的很明白了，定义如下：

The LLVM Pass Framework is an important part of the LLVM system, because LLVM passes are where most of the interesting parts of the compiler exist. Passes perform the transformations and optimizations that make up the compiler, they build the analysis results that are used by these transformations, and they are, above all, a structuring technique for compiler code.

接下来，了解如何写出一个Pass，以及其中的各个结构，这一点在官方的Docs上也有一篇介绍：[Writing An LLVM Pass](#)

由于篇幅较长，在此不再赘述，不同点在于以上实例实现了一个Function Pass，不过结构上与我们这次要写的Module Pass相同。

现在，来考虑程序的具体实现。首先需要知道Module、Function、BasicBlock的一些基本关系：

Module::iterator – Modules are translation units – Iterates through functions in the module

Function::iterator – Iterates through a function's basic blocks

BasicBlock::iterator – Iterates through instructions in a block

在[CMU的课件](#)中，发现了这样一个有趣的Tip：

● Use ++i rather than i++ and pre-compute the end

Avoid problems with iterators doing unexpected things while you are iterating – Especially for fancier iterators

下面列出几个重要的函数，在程序中可以使用到：

```
1 //Module.h
2     /// Get a short "name" for the module.
3     ///
4     /// This is useful for debugging or logging. It is essentially a
convenience
5     /// wrapper around getModuleIdentifier().
6    StringRef getName() const { return ModuleID; }
7
8 //Value.h
9     /// Return a constant reference to the value's name.
10    ///
11    /// This guaranteed to return the same reference as long as the value is
not
12    /// modified. If the value has a name, this does a hashtable lookup, so
it's
13    /// not free.
14    StringRef getName() const;
15
16 //BasicBlock.h
17    /// Returns the terminator instruction if the block is well formed or null
18    /// if the block is not well formed.
19    const Instruction *getTerminator() const LLVM_READONLY;
20    Instruction *getTerminator() {
21        return const_cast<Instruction *>(
22            static_cast<const BasicBlock *>(this)->getTerminator());
23    }
24
```

```

25 //Instructions.h
26 /// Return the specified successor. This instruction must be a terminator.
27 BasicBlock *getSuccessor(unsigned Idx) const;

```

这样，稍稍理解了LLVM的核心数据结构，剩下的工作就很简单了。

最终代码&运行结果

```

1 //MyPass.cpp
2 #include <queue> //bfs
3 #include <unordered_set> //mark visited BBs
4
5 #include "llvm/IR/CFG.h"
6 #include "llvm/IR/Function.h"
7 #include "llvm/IR/Instructions.h"
8 #include "llvm/IR/Intrinsics.h"
9 #include "llvm/IR/LegacyPassManager.h"
10 #include "llvm/IR/Module.h"
11 #include "llvm/Pass.h"
12 #include "llvm/Support/raw_ostream.h"
13 #include "llvm/Transforms/IPO/PassManagerBuilder.h"
14
15 using namespace llvm;
16
17 namespace {
18
19 class MyPass : public ModulePass {
20 public:
21     static char ID;
22     std::unordered_set<void*> st;
23     MyPass() : ModulePass(ID) {}
24
25     void visitBasicBlock(BasicBlock &bb) { //访问BasicBlock并打标记
26         if (st.count(&bb))
27             return;
28         else
29             st.insert(&bb);
30         errs() << "      # BasicBlock:" << &bb << '\n';
31         errs() << "      > Successors:\n";
32         auto termInst = bb.getTerminator();
33         int numSucc = termInst->getNumSuccessors();
34         for (int i = 0; i < numSucc; ++i) {
35             BasicBlock &cur = *termInst->getSuccessor(i);
36             errs() << "      : " << &cur << '\n';
37         }
38         errs() << "      > Predecessors:\n";

```

```

39         for (auto it = pred_begin(&bb), ed = pred_end(&bb); it != ed;
++it) {
40             BasicBlock *cur = *it;
41             errs() << "          : " << cur << '\n';
42         }
43         errs() << "          > Instructions:\n";
44         for (auto it = bb.begin(); it != bb.end(); ++it) {
45             Instruction *ii = *it;
46             errs() << "          : " << *ii << "\n";
47         }
48     }
49
50     void dfsBasicBlock(BasicBlock &bb) {
51         visitBasicBlock(bb);
52         auto termInst = bb.getTerminator();
53         int numSucc = termInst->getNumSuccessors();
54         for (int i = 0; i < numSucc; ++i) {
55             BasicBlock &cur = *termInst->getSuccessor(i);
56             dfsBasicBlock(cur);
57         }
58     }
59
60     void dfsFunction(Function &f) {
61         errs() << " + Function(DFS):" << f.getName() << '\n';
62         st.clear();
63         if (f.empty()) {
64             errs() << "    # Empty Function. Skipping.\n";
65             return;
66         }
67         dfsBasicBlock(f.getEntryBlock());
68     }
69
70     void bfsFunction(Function &f) {
71         errs() << " + Function(BFS):" << f.getName() << '\n';
72         if (f.empty()) {
73             errs() << "    # Empty Function. Skipping.\n";
74             return;
75         }
76         st.clear();
77         std::queue<BasicBlock *> q;
78         q.push(&f.getEntryBlock());
79         while (!q.empty()) {
80             if (st.count(q.front())) {
81                 q.pop();
82                 continue;
83             }
84             BasicBlock &cur = *q.front();
85             q.pop();
86             visitBasicBlock(cur);
87             auto termInst = cur.getTerminator();

```

```

87         int numSucc = termInst->getNumSuccessors();
88         for (int i = 0; i < numSucc; ++i)
89             q.push(termInst->getSuccessor(i));
90     }
91 }
92
93 virtual bool runOnModule(Module &M) {
94     errs() << "Module:" << M.getName() << '\n';
95     for (auto iter = M.begin(); iter != M.end(); iter++) {
96         Function &F = *iter;
97         dfsFunction(F);
98         bfsFunction(F);
99     }
100     return false;
101 }
102 };
103 } // namespace
104
105 char MyPass::ID = 0;
106
107 static void registerMyPass(const PassManagerBuilder &PMB,
108     legacy::PassManagerBase &PM) {
109     PM.add(new MyPass());
110 }
111
112 // works with "-O0" or no optimization options
113 static RegisterStandardPasses
114 RegisterMyPass_OPT0(PassManagerBuilder::EP_EnabledOnOptLevel0,
115     registerMyPass);
116
117 // works with "-O1", "-O2", ...
118 static RegisterStandardPasses
119 RegisterMyPass_OPT(PassManagerBuilder::EP_ModuleOptimizerEarly,
120     registerMyPass);

```

对应输出(DFS+BFS, 好长啊):

```

1 Module:TestMe.c
2   + Function(DFS):TestMe
3     # BasicBlock:0x5582f5a3dcb0
4       > Successors:
5         : 0x5582f5a3cf20
6         : 0x5582f5a3d350
7       > Predecessors:
8       > Instructions:
9         : %3 = alloca i32, align 4
10        : %4 = alloca i16, align 2
11        : %5 = alloca i32, align 4

```

```

12         : %6 = alloca i32, align 4
13         : %7 = alloca i32, align 4
14         : store i32 %0, i32* %3, align 4
15         : store i16 %1, i16* %4, align 2
16         : %8 = load i32, i32* %3, align 4
17         : %9 = icmp sgt i32 %8, 10
18         : br i1 %9, label %10, label %26
19 # BasicBlock:0x5582f5a3cf20
20     > Successors:
21         : 0x5582f5a3f080
22         : 0x5582f5a3f120
23         : 0x5582f5a3f760
24     > Predecessors:
25         : 0x5582f5a3dcb0
26     > Instructions:
27         : %11 = load i32, i32* %3, align 4
28         : %12 = load i16, i16* %4, align 2
29         : %13 = sext i16 %12 to i32
30         : %14 = mul nsw i32 %11, %13
31         : store i32 %14, i32* %5, align 4
32         : %15 = load i32, i32* %5, align 4
33         : switch i32 %15, label %22 [
34 i32 3, label %16
35 i32 5, label %20
36 ]
37 # BasicBlock:0x5582f5a3f080
38     > Successors:
39         : 0x5582f5a3efd0
40     > Predecessors:
41         : 0x5582f5a3cf20
42     > Instructions:
43         : %23 = load i32, i32* %5, align 4
44         : %24 = srem i32 %23, 7
45         : store i32 %24, i32* %6, align 4
46         : br label %25
47 # BasicBlock:0x5582f5a3efd0
48     > Successors:
49         : 0x5582f5a3d2b0
50     > Predecessors:
51         : 0x5582f5a3f080
52         : 0x5582f5a3f760
53         : 0x5582f5a3f120
54     > Instructions:
55         : br label %36
56 # BasicBlock:0x5582f5a3d2b0
57     > Successors:
58         : 0x5582f5a40520
59         : 0x5582f5a405c0
60     > Predecessors:

```

```

61         : 0x5582f5a40090
62         : 0x5582f5a3efd0
63     > Instructions:
64         : %37 = load i32, i32* %5, align 4
65         : %38 = load i32, i32* %6, align 4
66         : %39 = call i32 @test(i32 %37, i32 %38)
67         : %40 = icmp ne i32 %39, 0
68         : br i1 %40, label %41, label %44
69 # BasicBlock:0x5582f5a40520
70     > Successors:
71         : 0x5582f5a40570
72     > Predecessors:
73         : 0x5582f5a3d2b0
74     > Instructions:
75         : %42 = load i32, i32* %5, align 4
76         : %43 = call i32 @foo(i32 %42)
77         : store i32 %43, i32* %7, align 4
78         : br label %47
79 # BasicBlock:0x5582f5a40570
80     > Successors:
81     > Predecessors:
82         : 0x5582f5a405c0
83         : 0x5582f5a40520
84     > Instructions:
85         : %48 = load i32, i32* %7, align 4
86         : ret i32 %48
87 # BasicBlock:0x5582f5a405c0
88     > Successors:
89         : 0x5582f5a40570
90     > Predecessors:
91         : 0x5582f5a3d2b0
92     > Instructions:
93         : %45 = load i32, i32* %6, align 4
94         : %46 = call i32 @bar(i32 %45)
95         : store i32 %46, i32* %7, align 4
96         : br label %47
97 # BasicBlock:0x5582f5a3f120
98     > Successors:
99         : 0x5582f5a3efd0
100    > Predecessors:
101        : 0x5582f5a3cf20
102    > Instructions:
103        : %17 = load i16, i16* %4, align 2
104        : %18 = sext i16 %17 to i32
105        : %19 = call i32 @xfunc(i32 %18)
106        : store i32 %19, i32* %6, align 4
107        : br label %25
108 # BasicBlock:0x5582f5a3f760
109    > Successors:

```



```

110         : 0x5582f5a3efd0
111     > Predecessors:
112         : 0x5582f5a3cf20
113     > Instructions:
114         : %21 = call i32 (...) @yfunc()
115         : store i32 %21, i32* %6, align 4
116         : br label %25
117 # BasicBlock:0x5582f5a3d350
118     > Successors:
119         : 0x5582f5a3fc20
120         : 0x5582f5a40090
121     > Predecessors:
122         : 0x5582f5a3dcb0
123     > Instructions:
124         : %27 = load i16, i16* %4, align 2
125         : %28 = sext i16 %27 to i32
126         : %29 = icmp sgt i32 %28, 0
127         : br i1 %29, label %30, label %35
128 # BasicBlock:0x5582f5a3fc20
129     > Successors:
130         : 0x5582f5a40090
131     > Predecessors:
132         : 0x5582f5a3d350
133     > Instructions:
134         : %31 = load i16, i16* %4, align 2
135         : %32 = sext i16 %31 to i32
136         : %33 = shl i32 %32, 3
137         : %34 = sdiv i32 %33, 11
138         : store i32 %34, i32* %6, align 4
139         : br label %35
140 # BasicBlock:0x5582f5a40090
141     > Successors:
142         : 0x5582f5a3d2b0
143     > Predecessors:
144         : 0x5582f5a3fc20
145         : 0x5582f5a3d350
146     > Instructions:
147         : br label %36
148 + Function(BFS):TestMe
149     # BasicBlock:0x5582f5a3dcb0
150     > Successors:
151         : 0x5582f5a3cf20
152         : 0x5582f5a3d350
153     > Predecessors:
154     > Instructions:
155         : %3 = alloca i32, align 4
156         : %4 = alloca i16, align 2
157         : %5 = alloca i32, align 4
158         : %6 = alloca i32, align 4

```

```

159         : %7 = alloca i32, align 4
160         : store i32 %0, i32* %3, align 4
161         : store i16 %1, i16* %4, align 2
162         : %8 = load i32, i32* %3, align 4
163         : %9 = icmp sgt i32 %8, 10
164         : br i1 %9, label %10, label %26
165     # BasicBlock:0x5582f5a3cf20
166     > Successors:
167         : 0x5582f5a3f080
168         : 0x5582f5a3f120
169         : 0x5582f5a3f760
170     > Predecessors:
171         : 0x5582f5a3dcb0
172     > Instructions:
173         : %11 = load i32, i32* %3, align 4
174         : %12 = load i16, i16* %4, align 2
175         : %13 = sext i16 %12 to i32
176         : %14 = mul nsw i32 %11, %13
177         : store i32 %14, i32* %5, align 4
178         : %15 = load i32, i32* %5, align 4
179         : switch i32 %15, label %22 [
180     i32 3, label %16
181     i32 5, label %20
182 ]
183     # BasicBlock:0x5582f5a3d350
184     > Successors:
185         : 0x5582f5a3fc20
186         : 0x5582f5a40090
187     > Predecessors:
188         : 0x5582f5a3dcb0
189     > Instructions:
190         : %27 = load i16, i16* %4, align 2
191         : %28 = sext i16 %27 to i32
192         : %29 = icmp sgt i32 %28, 0
193         : br i1 %29, label %30, label %35
194     # BasicBlock:0x5582f5a3f080
195     > Successors:
196         : 0x5582f5a3efd0
197     > Predecessors:
198         : 0x5582f5a3cf20
199     > Instructions:
200         : %23 = load i32, i32* %5, align 4
201         : %24 = srem i32 %23, 7
202         : store i32 %24, i32* %6, align 4
203         : br label %25
204     # BasicBlock:0x5582f5a3f120
205     > Successors:
206         : 0x5582f5a3efd0
207     > Predecessors:

```

```

208         : 0x5582f5a3cf20
209     > Instructions:
210         : %17 = load i16, i16* %4, align 2
211         : %18 = sext i16 %17 to i32
212         : %19 = call i32 @xfunc(i32 %18)
213         : store i32 %19, i32* %6, align 4
214         : br label %25
215 # BasicBlock:0x5582f5a3f760
216     > Successors:
217         : 0x5582f5a3efd0
218     > Predecessors:
219         : 0x5582f5a3cf20
220     > Instructions:
221         : %21 = call i32 (...) @yfunc()
222         : store i32 %21, i32* %6, align 4
223         : br label %25
224 # BasicBlock:0x5582f5a3fc20
225     > Successors:
226         : 0x5582f5a40090
227     > Predecessors:
228         : 0x5582f5a3d350
229     > Instructions:
230         : %31 = load i16, i16* %4, align 2
231         : %32 = sext i16 %31 to i32
232         : %33 = shl i32 %32, 3
233         : %34 = sdiv i32 %33, 11
234         : store i32 %34, i32* %6, align 4
235         : br label %35
236 # BasicBlock:0x5582f5a40090
237     > Successors:
238         : 0x5582f5a3d2b0
239     > Predecessors:
240         : 0x5582f5a3fc20
241         : 0x5582f5a3d350
242     > Instructions:
243         : br label %36
244 # BasicBlock:0x5582f5a3efd0
245     > Successors:
246         : 0x5582f5a3d2b0
247     > Predecessors:
248         : 0x5582f5a3f080
249         : 0x5582f5a3f760
250         : 0x5582f5a3f120
251     > Instructions:
252         : br label %36
253 # BasicBlock:0x5582f5a3d2b0
254     > Successors:
255         : 0x5582f5a40520
256         : 0x5582f5a405c0

```

```

257         > Predecessors:
258             : 0x5582f5a40090
259             : 0x5582f5a3efd0
260         > Instructions:
261             : %37 = load i32, i32* %5, align 4
262             : %38 = load i32, i32* %6, align 4
263             : %39 = call i32 @test(i32 %37, i32 %38)
264             : %40 = icmp ne i32 %39, 0
265             : br i1 %40, label %41, label %44
266     # BasicBlock:0x5582f5a40520
267         > Successors:
268             : 0x5582f5a40570
269         > Predecessors:
270             : 0x5582f5a3d2b0
271         > Instructions:
272             : %42 = load i32, i32* %5, align 4
273             : %43 = call i32 @foo(i32 %42)
274             : store i32 %43, i32* %7, align 4
275             : br label %47
276     # BasicBlock:0x5582f5a405c0
277         > Successors:
278             : 0x5582f5a40570
279         > Predecessors:
280             : 0x5582f5a3d2b0
281         > Instructions:
282             : %45 = load i32, i32* %6, align 4
283             : %46 = call i32 @bar(i32 %45)
284             : store i32 %46, i32* %7, align 4
285             : br label %47
286     # BasicBlock:0x5582f5a40570
287         > Successors:
288         > Predecessors:
289             : 0x5582f5a405c0
290             : 0x5582f5a40520
291         > Instructions:
292             : %48 = load i32, i32* %7, align 4
293             : ret i32 %48
294 + Function(DFS):xfunc
295     # Empty Function. Skipping.
296 + Function(BFS):xfunc
297     # Empty Function. Skipping.
298 + Function(DFS):yfunc
299     # Empty Function. Skipping.
300 + Function(BFS):yfunc
301     # Empty Function. Skipping.
302 + Function(DFS):test
303     # Empty Function. Skipping.
304 + Function(BFS):test
305     # Empty Function. Skipping.

```

```
306 + Function(DFS):foo
307     # Empty Function. Skipping.
308 + Function(BFS):foo
309     # Empty Function. Skipping.
310 + Function(DFS):bar
311     # Empty Function. Skipping.
312 + Function(BFS):bar
313     # Empty Function. Skipping.
```