C到LLVM编译器简单实现

和杨梁

李天逸

马浩宇

何志辉

1. 实验环境及配置方法

- 系统：`Ubuntu 18.04`
- 语言：`Python`
- 工具：`antlr4`；`antlr4-python3-runtime`；`llvmlite`

  安装antlr4；
  安装antlr4-python3-runtime、llvmlite；

  ```
  pip install antlr4-python3-runtime
  pip install llvmlite
  ```

2. 运行说明

注意以下命令运行路径为`/src`目录下，即与`main.py`同级

生成某个源代码的 LLVM IR 代码

```
python main.py test/xxx.c    #该命令会在源代码同级目录下生成xxx.ll文件
```

运行LLVM IR代码

```
lli xxx.ll
```

3. 代码结构

Generator

> Generator.py 实现由 C 代码转到 LLVM IR 代码
>
> SymbolTable.py 实现了符号表

Parser

> simpleC.g4 C语言部分文法实现
>
> simpleC.interp 本项及以下5项由antlr4自动生成
>
> simpleC.tokens
>
> simpleCLexer.interp
>
> simpleCLexer.py
>
> simpleCLexer.tokens

> simpleCListener.py
>
> simpleCParser.py 用于语法分析
>
> simpleCVisitor.py 语义分析基于ANTLR的Visitor模式进行

test

> bubblesort.c 实现了冒泡排序，可验证分支选择、循环和数组的正确处理
>
> calculator.c 实现了栈结构的四则运算，可验证自定义函数、复杂逻辑的正确处理

4. 具体实现

4.1 词法语法分析实现

词法与语法分析的实现借助了 `antlr4` 工具，同时使用了 `antlr4-python3-runtime` 工具使得生成的词法语法分析代码为 `Python` 代码。

词法分析的规则定义在 `SimpleCItemLexer.g4` 中，其中定义了库名 `LIB`、变量名 `ID`，`INT` 和 `DOUBLE`，`CHAR`，`STRING` 类型的立即数 `INT`，`DOUBLE`，`CHAR`，`STRING`。`OPERATOR` 中定义了运算符 （`+-*/%!`） 与

`== != < > <= >=`，在 `CONJUNCTION` 中定义了逻辑判断的连接符 `&&` 和 `||`。定义了行注释、多行注释以及空格缩进符换行符等字符的跳过操作。

语法分析的规则定义在了 `SimpleC.g4` 中，入口开始符号是 `program`，产生式为 `program-> (include)* (defineBlock| myFunction)*`，在 `include` 中定义了调库操作， `defineBlock` 是对各种变量的声明，`myFunction` 则是对函数的定义。`block` 中定义了各种语句，包含声明、赋值、if、while、for、return 语句。

`expr` 中则是对表达式的定义。余下的细节部分可以参考具体代码 `simpleC.g4` 和 `simpleCItemLexer.g4`。

4.2 函数定义与调用的实现

```
llvm_type = ir.FunctionType(self.visit(ctx.getChild(0)), type_list)
llvm_func = ir.Function(self.module, llvm_type, name = func_name)
```

上面是定义函数需要用到的语句，函数类型由其返回值和变量数目和类型决定。

函数参数存储需要符号表进一层，因为函数参数的作用域等价于函数内定义的变量，然后遍历获取函数参数列表即可，出参数时需要将符号表退一层。

之后是函数调用需要用到的语句，需要先读取 LLVM 函数本身和其参数列表。

```
builder.call(func, para_list)
```

此外，我们还实现了 printf 和 scanf 函数的调用，调用方法与 C 标准库基本一致。

4.3 程序的分支选择和循环结构

4.3.1 程序的分支选择

支持C中的if-else和if-elseif-else分支选择结构。在解析分支选择的过程中，以if-elseif-else为例，先将分支选择结构分解为分支部分(if_block)和分支之后的部分(endif_block)。对于分支部分解析if condition部分的语句，然后根据condition部分执行的结果决定分支到if下方的语句(true_block)还是elseif下方的语句(false_block)，然后对elseif-else重复这一过程，最后解析分支之后的部分。

4.3.2 程序的循环结构

支持C中的while和for两种循环结构。

对于while结构，和if-else结构类似，先解析while condition部分的语句(cond_block)，然后根据condition部分执行的结果决定分支到while内部下方的语句(body_block)还是while结束之后的语句(endwhile_block)，每完成一轮body_block的执行就分支到cond_block。

对于for结构，与while结构基本完全一致，只是在解析for结构前需要先解析Define Sentence部分，在每次body_block解析后需要解析Iterator部分，然后分支到cond_block。

4.4 符号表的实现

在 SymbolTable.py 中实现了 SymbolTable 类，用于管理变量。

符号表主要由一个字典数组dic_list组成，数组的每个元素代表一层，每个 key 是变量在C中的名字（ID），value是LLVM的类型和名称等信息，其次还有一个变量cur_scope记录当前访问的层数。当进入一个新的作用域时，append一个dict到list末尾，退出后则直接删除。在作用域内新增变量时，遍历list最后一个dict的所有key值，重复则报错，否则就新建键值对。当需要查找当前曾层数的变量时，由内层（即list开头）从前到后访问，就可以逐层找变量，找不到就产生异常。而对于全局变量，使用函数is_global来进行判断，若当前list只有一层，就代表在全局位置。

5. 参考资料

参考了以下链接的资料：关于ANTLR 的使用及Python-ANTLR

https://decaf-lang.github.io/minidecaf-tutorial/docs/lab1/antlr.html

https://decaf-lang.github.io/minidecaf-tutorial/docs/ref/python-dzy.html