Pyscal 编译器 实验报告

胡浦云 2018202133

实验目标

实现 PL0 编译器及拓展语法

实验思路

1. 拓展语法
    1. 类型系统
    2. 数组
    3. 字符串
    4. 函数引用
    5. break 语句（参见 `loop.pys`）
2. 目标平台选择
    1. 由于 PL0 函数可以嵌套，为实现函数引用需实现闭包，考虑使用有 gc 的目标平台。
    2. 在 JVM 和 Python 之间选择了 Python，从而该实现将类 PL0 语言编译为 Python bytecode，故名 Pyscal

代码结构

编译器

将 Pyscal 代码编译为 `S-expression` 格式 python 字节码

1. pl0.l
   flex 源文件

2. pl0.y.in pl0.y
   bison 源文件

3. ast.cpp ast.h
   抽象语法树

4. pl0_process.h
   建立抽象语法树

5. block.cpp block.h
   维护代码结构，一个函数称为一个 block，储存其符号表及常量表；
   以及类型系统，对类型进行 `mangle` 和 `demangle`。例如：一个 接受两个 `int` 作为参数的函数，该函数返回一个函数，这个函数接受一个 `real` 和一个 `string` 作为参数，返回一个函数，这个函数接受一个 `int[1..2][5..7]` 数组作为参数，无返回值 表示为 `iirsi[1.2[5.7n(1(2(2`

6. instruction.cpp instruction.h
   从 ast 生成Python bytecode

7. global.cpp global.h
   定义的全局函数

8. resolve.cpp
   建立符号表

9. util.h
   黑魔法

汇编器

简单的 lisp 解释器，解释执行 `S-expression` 并通过内置函数生成 Python bytecode（`.pyc` 文件）

1. main.py
   将 `S-expression` 解释执行，生成 Python bytecode
2. lib.py
   内置函数

使用说明

编译

```
$ shopt -s extglob # 允许反选
$ python3 process.py pl0.y.in pl0.y pl0_process # 可选，生成 pl0.y 及 pl0_pro
$ flex pl0.l
$ bison pl0.y
$ gcc -o pyscal -O2 !(lex.yy).{cc,cpp}
```

使用

```
$ .\pyscal 2> bytecode.txt < target.pys
$ python3 sexp.py bytecode.txt
$ python3 .\a.pyc
```

由于 Python bytecode 格式在 3.6 3.8 各有一次改动，因而该程序可以在 3.6 3.7 正常运行，或者在 3.8+ 运行除循环以外部分。

> Written with StackEdit.