Rkt2Cpp 结题报告

杨子毅 1600011063

~~0.前言——关于程序打包：~~

~~由于(可能是Ubuntu的racket编译器的)一些bug,desugar的代码再ubuntu16.04和18.04均不能raco exe编译(会一直编译而不停下来)而debian上实测没问题。所以ubuntu大概需要用Drracket执行后将输入代码手动复制到窗口。C++的Format也没办法集成进去。~~找到了原因:ubuntu 下 raco 莫名其妙的 bug ,输入文件不能叫 desugar.rkt，改成后面加个数字就好了。（然而担心在其他测试平台上出现其他问题，所以只是加了删除线。）

1.选题背景：

学期初选了编译原理以及函数式程序设计，加上本身对 Programming Language 比较感兴趣，想要作一个语言转化的编译器。

2.选题介绍

Racket 作为新兴的一门基于 Scheme 的编程语言，兼具函数式语言的友好特性同时，社区活跃度较高，社区文档齐全，是一门有着优秀前景的语言(在 DSL 方面以及很受欢迎)。而 C++则是编程语言中的中坚力量，接近 40 年的历史依然在不断维护更新，也是得益于他的诸多优点——运行速度、特性齐全、功能完备。而自 C++11 以后，得益于函数式语言的发展，C++也在借鉴和吸收其中的特性（主要是泛型 template 和泛型 lambda）。

而 Rkt2Cpp 就是要将 Racket 中的部分核心特性运用编译的一些方法转化为 C++的特性（虽然并不完备）。

3.程序架构

借鉴了《Modern Compiler Design》上对 Haskell 编译的思想——将 Racket 语言的特性（语句）分为 Function Core 和 Sugar。顾名思义：Function Core 就是 Racket 语句中比较偏向 Lambda 演算的部分；而 Sugar 就是语法糖，可以通过转化（脱糖）转化为 Function Core 的形式。于是主程序 (Rkt2Cpp) 先经过 desugar 得到脱糖后的 racket 表达式，再通过 transcore 得到对应的 C++代码。其中主程序的转化主要是将语句分为表达式和返回语句（于是会出现对于返回 void 可能发生的一些错误）。

除了主程序，还有一部分功能是需要再 C++中实现一些 Racket 中的核心功能（如加减乘除再 racket 中是可以多参数的），由于 Racket 内置函数丰富，将 Racket 转化过去的语言实际上是需要完成一个 C++库（或者给出 Racket 中的实现）。在 Rkt2Cpp.rkt 中提供了这样的接口（(member (car exp) selfdefine)处），并加入了两个简单的函数实现（用于样例）。

~~总体使用方法：DrRacket 运行 Rkt2Cpp.rkt，输入要转化的程序，将输出粘贴到.cpp 文件并编译即可。（如果 raco 没问题可以，可以 raco exe Rkt2Cpp.rkt; Rkt2Cpp <input.rkt >output.cpp)~~

运行环境：g++7 以上或 clang5（确保生成的代码能编译）。Racket6.3 以上。Clang-format(Debian 系可直接在 apt 下载）。

运行方法：./run.sh $1 $2 其中两个参数分别为输入的 rkt 文件名及输出的 cpp 文件名。（其中 Rkt2Cpp 由命令 raco exe Rkt2Cpp.rkt 生成）

4.程序具体功能及实现

4.0：关于程序输入

由于时间精力有限，且作为一个课程项目而非工程项目，程序的输入仅为 racket 的一个子集，对输入有一些限制。

<prog> ::= <top>*

<top> ::= <def> | <definedvar>

<def> ::= (define (<var> <var>*) <body>)
| (define <var> <exp>)

<exp>::= <definedvar>
| <atomic>
| <prim>
| (lambda (<var>*) <body>)
| (let ((<var> <exp>)*) <body>)
| (letrec ((<var> <exp>)*) <body>)
| (cond (<exp> <exp>)* [(else <exp>)])
| (and <exp>*)
| (or <exp>*)
| (if <exp> <exp> [<exp>])
| (set! <var> <exp>)
| (begin <body>)
| (list <atomic>*)//

<body> ::= <top>* <exp>

语言描述：在输入主程序中的 s 表达式只能是两类，分别是 define 和调用 define 过的变量/函数（Racket 中的专业术语叫过程），及调用 define 过的变量/函数。其中变量对应<exp>的结果，而函数对应<body>的 procedure。一下对主程序中的做法进行简单的描述。

4.1:define——转为 class（若为过程定义则为 template class）

4.2:if——可能为语句也可能为返回值（甚至可能是 void）需要进行判断。而由于类型系统的原因，并不能很好的处理 void 的情况（第五节会单独提到）。

4.3:set!——同样可能为（void）的返回值，也可能是一个语句。

4.4:加减乘除(prim)——实现在 rkt2cpp.h 中

4.5:list——实现在 rkt2cpp.h 中，目前只支持 car 与 cdr 操作。

4.6:begin——begin 块整体必为返回语句，即可转化为前面一堆表达式加上最后一句的返回语句

4.7:lambda——直接转为 C++14 中的 generic lambda。

4.8:atomic——转化为 C++中对应类型的常量表达式。

4.9:definedvar——通过建立符号表（func-list 和 var-list）来判断调用方式。若调用位于主程序中（即外侧没有嵌套括号）则判断返回类型是否为 void。不的话输出该过程或变量的结果。

4.10:其他均为 lambda 演算衍生出的基本过程。通过 desugar2.rkt 的程序进行脱糖得到 lambda 表达式或 if 表达式等等。

一些测试样例见 test 文件夹。

5.项目的缺陷

5.1:void——在 C++中 void 不是一个数据类型，而 Racket 中 void 算是类型系统中的一种（可以用 lambda 表达）。于是一些函数是无法转化（编译）的。例：

```
(define (f x y) (if (< x y) x (void)))
```

5.2:动态类型系统与静态类型系统。这很可能是之前大多数工具/项目是用 C++实现 Racket 而非运用编译的原因（实际上，Rkt2Cpp 中也存在一些 C++实现 Racket 的部分）。

例：

```
(define a 1)
(define (f x) (set! x (list 1 2)))
(f a)
```

综上，出现问题的主要原因就是类型系统的差异。

6.总结

整个项目收获最大的就是思考的过程。整个项目的前几个星期一直在查阅各种书籍以及网页等资料，将各种 idea 综合到一起，并加入自己的想法，形成了整个项目的结构。之后基本都是自己在想各种东西如何与编译结合，如何实现转化，如何解决问题。其次也顺带巩固了一些编译原理和函数式语言的知识。