Scheme解释器(no-side-effect)实现报告

by 叶子豪

大一暑期PPCA自选作业（2015）

1 实现环境

Java语言，vim + openjdk

2 参考资料

r5rs标准，《计算机程序的构造和解释》（SICP）

3 实现过程

一开始我看了SICP中的4.1节元循环解释器后感觉框架过于复杂，于是自作主张地写了一套框架，但是在实现的过程中发现了无法支持的各式各样的问题。 不断改进之后， 我发现我的框架跟书上的几乎一样， SICP上的模型的确是最精简的模型。

代码一共重构了好几次， 每一次都比前一次短， 最后的版本Java代码一共957行， 预加载的Scheme代码一共59行。

4 类型设计

由于Scheme是弱类型语言，所以直接用Java的类型可能比较不方便，所以我用一个抽象类Type表示所有的类型，它有以下的虚函数：

```
abstract void display();

abstract boolean isNum();

abstract boolean isFun();

abstract boolean isBool();

abstract boolean isStr();

abstract boolean isChar();

abstract boolean isPair();

abstract boolean isNull();

abstract boolean isQuote();
```

（后面几个是为了方便由基类判断类型）

然后定义几种基本类型：

```
Number, Function, Str, Char, Pair, Null, Quote, Bool
```

均由Type继承而来，

对于每种类型，定义一个构造函数，根据其特性写一个display函数（其中Pair的display函数需要递归调用）

其它的类型比较简单，这里主要介绍以下Number和Function类型：

Number:

每个Number里面保存了一个有理数（分子分母用BigInteger来表示）和一个double类型的实数，还有一个标记Style表示这个数属于有理数还是实数。

构造函数分三类：由字符串构造，由两个BigInteger构造，由double构造。

Function:

每个函数我们保存它的每个参数名（String），它作用时的环境(Environment)，以及它的过程在程序块的位置[l，r]，其中参数中有'．'的情况特殊处理一下，把后面的参数放到一个list中即可。

5 词法分析

首先把程序中的'全部代替为(quote)（因为前者难以分析作用范围）

然后把字符序列转换为单词：

```
(define (id x) x)
```

to:

```
(
define
(
id
x
)
x
)
```

每个位于第i位的单词，我们记录一个match[i]表示i开始的过程结束的位置。

这样[i，match[i]]就表示一段完整的过程（比如(id x)）。

6 环境

Environment用一个类似于树的结构的东西储存，每个环境中要保存当前的变量列表和一个它的父环境，查询的时候如果在当前的变量列表中没有找到就去父环境中查找，整个过程是可持久化的，不改变任何环境，只构造新环境。一开始我的变量列表是用一个链表来实现的，找变量的过程中找最近插入的一个。后来改用了HashMap，但是速度并没有明显的提升。

7 求值部分

核心的函数不外乎eval和apply，一开始我把define，let这些都当作基本函数，所以我eval和apply函数是这样写的：

```
Type eval(int l, int r, Environment nowEnv);
Type apply(Environment prev, Function f, int l, int r);
```

其中apply的参数f表示当前执行的函数，[l，r]表示参数在程序中的位置，然后prev表示参数的求值环境。在apply里面对参数进行求值然后调用新的函数。

这样的话可以完成Scheme的大部分函数，但是apply和map函数却很难实现，因为我发现用我的框架很难把list变成一个参数表。后来经过思考，我发现我的框架非常的愚蠢，define和let，letrec这些没有必要当作基本函数可以放在eval里面执行，只需要修改一下apply函数的参数就可以了：

```
Type apply(Function func, ArrayList <Type> args);
```

args表示已经计算好的参数（在eval过程中计算，这样也省了一个参数prev），这样整个程序简洁了很多，apply和map函数也可以很愉快的实现了。（事实上这与SICP上的写法并无区别）

8 具体函数实现

由于有类型的设计,函数的实现还是比较方便的,类似与在Scheme中的定义。

其中+,-,*,/在实现的时候都是以其中一个不精确的变量为其类型。

equal?函数在实现的时候递归调用,精确到每一个具体基本变量。

eq?函数在实现的时候,对于Pair和Str直接比较地址是否相同,比较容易。

9 使用

新建一个test.scm文件保存需要运行的程序。

安装了JDK之后，在终端直接运行:

```
javac interpreter.java
java interpreter
```

即可。

10 问题

由于时间关系，很多具体的函数都没有实现，而且这个解释器在运行一些递归深度比较深的程序时会爆栈（比如八皇后搜到第7层时）,目前并不知道有什么比较好的解决方法。