

函数式语言程序设计2019课程作业报告

一、作者信息

- 姓名：洪方舟
- 学号：2016013259
- 班级：软件62

二、执行方法

1. 核心语言、代数数据类型

由于可以采取黑盒测试，因此没有显式的执行方法，主要体现在测试的调用。

2. Parser

进入项目文件夹后执行下面的语句

```
stack run
```

进入程序后输入 `parser` 按回车，再输入想要测试的语句后，按下回车后，下面会显示解析的AST，表达式的类型，以及表达式的结果。

3. REPL

进入项目文件夹后执行下面的语句

```
stack run
```

进入程序后输入 `repl` 按回车，就进入了交互式环境，具体的使用方法请参照下面的实现细节。

三、实现目标

1. 核心语言

实现文档中的所有有关要求

2. 代数数据类型

实现文档中的所有有关要求

3. 文法设计和parser

实现的文法中不包含ADT的部分，仅包含核心语言部分。

具体文法

数据类型的文法

- `bool` : 布尔类型
- `int` : 整型
- `char` : 字符
- `arrow <type> <type>` : 函数类型

表达式的文法

- `EBoolLit` : `true` / `false`
- `EIntLit` : 有符号整数
- `ECharLit` : 单引号包围的字符
- `ENot` : `not <expr>`
- `EAnd` : `<expr> and <expr>`
- `EOr` : `<expr> or <expr>`
- `EAdd` : `<expr> + <expr>`
- `ESub` : `<expr> - <expr>`
- `EMul` : `<expr> * <expr>`
- `EDiv` : `<expr> / <expr>`
- `EMod` : `<expr> % <expr>`
- `EEq` : `<expr> = <expr>`
- `ENeg` : `<expr> != <expr>`
- `ELt` : `<expr> < <expr>`
- `EGt` : `<expr> > <expr>`
- `ELe` : `<expr> <= <expr>`
- `EGe` : `<expr> >= <expr>`
- `EIf` : `if <expr> then <expr> else <expr>`
- `ELambda` :
 - 文法: `lambda <type> => <identifier> -> <expr>`
 - 对应的AST: `ELambda (<identifier>, <type>) <expr>`
- `ELet` :
 - 文法: `let <identifier> := <expr1> in <expr2>`
 - 对应的AST: `ELet (<identifier>, <expr1>) <expr2>`
- `ELetRec` :
 - 文法: `function <type1> <identifier1> (<type2>, <identifier2>) {<expr1>} in <expr2>`
 - 对应的AST: `ELetRec <identifier1> (<identifier2>, <type2>) (<expr1>, <type1>) <expr2>`

- `EVar` :
 - 文法: `<identifier>`
 - 对应的AST: `EVar <identifier>`
- `EApply` :
 - 文法: `apply <expr1> to <expr2>`
 - 对应的AST: `EApply <expr1> <expr2>`

4. REPL

下面给出REPL的实现的功能及使用方法

绑定变量

- 文法: `bind <identifier> := <expr>`
- 这样做就将 `<expr>` 绑定到了 `<identifier>` 上, 下面的表达式就可以使用这个 `<identifier>`

查看类型

- 文法: `:t <expr>`
- 这样做将会输出 `<expr>` 的类型

查看表达式的值

- 文法: `<expr>`
- 这样做将会输出 `<expr>` 的值

退出交互式环境

- 文法: `:q`
- 这样做将会退出交互式环境

清除上下文

- 文法: `:c`
- 这样做将会清除上面绑定的变量

四、作业的思路及亮点

1. EvalType

这个部分较为简单, 只需要维护一个上下文栈, 栈中存放当前上下文中变量的绑定类型。

2. EvalValue

这个部分也较为简单, 与EvalType差不多, 维护一个上下文栈, 栈中存放当前上下文变量绑定的表达式, 同时用一个结构体存储函数部分应用的结果。

虽然文档中并没有明确说明需要支持惰性求值, 测试用例中也没有相关测例, 但是该部分还是支持了该特性。

3. ADT

这个部分只需要将ADT的构造函数当成普通函数来处理即可。

4. Parser

Parser部分使用了megapasec库辅助实现。实现起来也没有什么困难，只要按照自定义的文法，对于每一条文法规则写解析函数。

在处理中缀表达式的时候有一些困难，因为要实现中缀表达式，该文法就包含了左递归，而megapasec支持的为LL(1)文法，为此本需要消除左递归，但是megapasec的 `makeExprParser` 函数可以帮助我们消除左递归。

5. REPL

在parser的基础上，REPL就较为容易实现，只需要维护一个上下文的绑定即可。

五、参考资料

- [Megaparsec tutorial from IH book](#)
- [Parsing a simple imperative language](#)