PA1-A 实验报告

工作内容

PA1-A 实验要求使用 `lalr1` 添加三个新语法的支持：抽象类、局部类型推断和 First-class functions，需要做这几个方面的修改：

1. 修改 parser 中的产生式，使得它可以支持新的语法
2. 修改 ast 中的结构，把新的信息保存下来
3. 修改其他代码以适配 ast 的修改

抽象类

需要添加新的产生式：

```
1  ClassDef -> Abstract Class Id MaybeExtends LBrc FieldList RBrc
2  FuncDef -> Abstract Type Id LPar VarDefListOrEmpty RPar Semi
```

分别处理抽象类和抽象成员函数。然后在 ast 的 `ClassDef` 中添加 `abstract_: bool`，并允许 `FuncDef` 的 `body` 为空。

局部类型推断

需要添加新的产生式：

```
1   Simple -> Var Id Assign Expr
```

同时添加 `Var` 的类型，在打印时直接输出 `<none>` 。

First-class functions

匿名函数

添加产生式：

```
1  Expr -> Fun LPar VarDefListOrEmpty RPar Rocket Expr
2  Expr -> Fun LPar VarDefListOrEmpty RPar Block
```

为 `Expr` 添加新的 `Lambda` 类型，记录下参数和 `body` 。

函数类型

添加产生式：

```
Type -> Type LPar TypeList RPar
TypeList -> TypeList Comma Type
TypeList -> Type
TypeList ->
```

为 `SynTy` 添加新的 `Lambda` 类型，然后记录下参数类型。

调用语法

添加产生式：

```
Expr -> Expr LPar ExprListOrEmpty RPar
```

需要解决 `shift-reduce` 冲突，因为这里也有和 Dangling Else 类似的问题： `map(func)(1)` (见于 testcase/S1/lambda8.decaf)，此时可以理解成 `(map(func))(1)` 也可以理解成 `map((func)(1))`，设置优先级后保证解析为 `(map(func))(1)` 。

问题回答

Q1. 有一部分 AST 结点是枚举类型。若结点 `B` 是枚举类型，结点 `A` 是它的一个 variant，那么语法上会不会 `A` 和 `B` 有什么关系？限用 100 字符内一句话说明。

语法上会对应 `A ::= B` 的产生式。如 `enum StmtKind` 有 `If` `While` `Return` 等等 variant，都对应了 `stmt` 中的 `if` `while` `return` 开头的产生式。

Q2. 原有框架是如何解决空悬 else (dangling-else) 问题的？限用 100 字符内说明。

`lalr1` 会汇报 `warning: Conflict at prod` MaybeElse -> Else Blocked *and* MaybeElse -> , `both's PS contains term` Else . 冲突错误，按照语法要求， `else` 与最近的 `if` 匹配，所以这里选择 shift 而不是 reduce 来解决冲突。

Q3. PA1-A 在概念上，如下图所示：

```
作为输入的程序（字符串）
    --> lexer --> 单词流（token stream）
    --> parser --> 具体语法树（CST）
    --> 一通操作 --> 抽象语法树（AST）
```

输入程序 lex 完得到一个终结符序列，然后构建出具体语法树，最后从具体语法树构建抽象语法树。 这个概念模型与框架的实现有什么区别？我们的具体语法树在哪里？限用 120 字符内说明。

框架里没有显式地构造一颗 CST 出来，但 CST 是记录在程序的执行过程里，代码相当于在遍历 CST 结点，然后插入的模板代码，进行一通操作，直接构造 AST 的结点，跳过了 CST 的构造。