介绍 / 亮点

本次实验由三个部分组成:设计 `AST` , 实现 `SimpleLALR(2)` 自动机的生成,实现 `SimpleLALR(2)` 语法分析算法。

1. 因为`LALR(1)`不能满足给定的文法，于是基于其提出`SimpleLALR(2)`算法。
2. 完成泛用性较高的`LALR(1)` 自动机的生成代码，`LALR(1)`自动机是`SLALR(2)`的基础。
3. 实现 `SLALR(2)` 语法分析算法，它是`LALR(1)`的一个简单升级版。
4. 设计实现抽象语法树，设计了一种简洁的输出方式。

详情请见附件`语法分析设计方案-SimpleLALR2方法与抽象语法树.pdf`。

如何运行 / 测试

应为本次实验用`javascript`实现，它是解释性语言，所以本身没有可执行文件。

你可以选择`node mjava-parser.js test.txt`来运行，`test.txt`是测试文件，里面应该含有`mjava`源代码。

程序会将正常的`AST`输出至本目录下的`syntaxOut.txt`文件和标准输出。 当有报错的情况，将输出至错误输出。

其他问题

1. 抽象语法树的定义，见`语法分析设计方案-SimpleLR2方法与抽象语法树.pdf`。
2. `SLALR(2)`自动机的生成代码是`builder.js`, `mjava`的语法分析代码是`mjava-parser.js`。
3. 为什么不直接将`tokenOut.txt`作为输入？ 因为本实验的程序会提供友好的报错信息，这部分功能需要源代码。 原理上完全可以将词素序列作为输入。 详情见`mjava-parser.js`的第9行`getToken`函数。
4. 接问题3, 本程序依赖实验一的词法分析程序，本人已经将源程序和依赖的`jar`包放好了位置。因此只需要nodejs和java环境即可运行。