****



**C--编译器前端设计报告**

**——《编译原理与技术》课程大作业**

**姓名 吴显宗 刘宏伟 吴金垚 邱胜慧**

**学号 3021208007 3020205094**

**3522074811 3522074801**

**学院名称 智能与计算学部**

**专 业 网络空间与安全**

### 1 前言

老师好，我们是第二组，所有同学都来自2021级网络空间安全1班。作为刚刚转专业来到智算学部的大二同学，个人能力实在有限，所以只能通过查阅教材和参考前人的思路完成这次大作业，若有问题还请老师批评指正！在展示环节中，老师的评价“还挺好”是对我们几天以来努力的最大肯定，也使我们收获了满满的成就感。

在词法分析器中，我们能够将源代码分解为一个个具有意义的词法单元。NFA自动构建将正则表达式转换为NFAε-NFA，DFA确定化将NFAε-NFA转换为确定的DFA，DFA最小化减少DFA的状态数目，配置读入和Token匹配进行状态转移和匹配操作。这些步骤共同实现了词法分析器的功能。

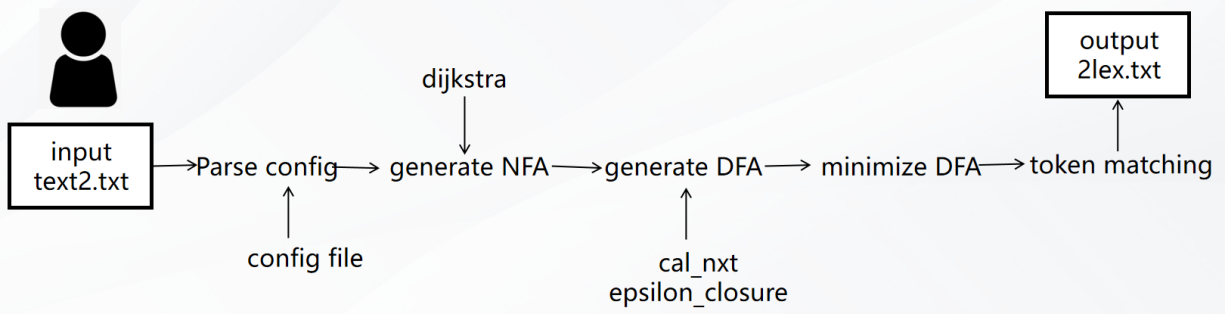
在语法分析器中，首先读入给定的C--文法grammar，计算FIRST集和FOLLOW集，然后安装课本中讲授的算法计算闭包与GO函数求LR(0)项目集规范族并填充SLR(1)分析表的GOTO部分，然后构造SLR(1)分析表的ACTION部分，然后读入词法分析器的输出结果lex.txt，根据SLR(1)分析表以及token流进行移进归约分析。这些步骤共同实现了语法分析器的功能。

### 2 有关编译步骤的说明

请老师打开final-src文件夹，测试样例输入文件名为test2.txt，在这个文件目录下分别先后编译lexical.py和syntax.py文件，可以得到2lex.txt和2gra.txt文件，即词法分析器和语法分析器的输出。

### 3 词法分析器描述

#### 3.1 流程图



用户输入数据后，先解析config配置文件，然后生成NFA 再经历一个NFA转DFA的过程，再把DFA最小化，最后是符号匹配，作用是判断当前dfa的状态是否为中止态。此外在代码中我们加了大量注释，有助于对代码架构的理解。

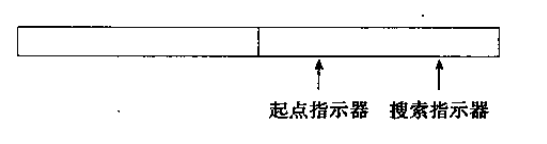
#### 3.2 符号表

我们定义了一个config文件，里面包括了所有关键字、运算符、界符、标识符和整数的定义。

使用分隔符“~”切分每行，得到类型名、正则表达式和序号。

其中采用了INT~$(非零数$+所有数$\*$)$ | $(-$+非零数$+所有数$\*$)~INT的方式定义了正整数和负整数。

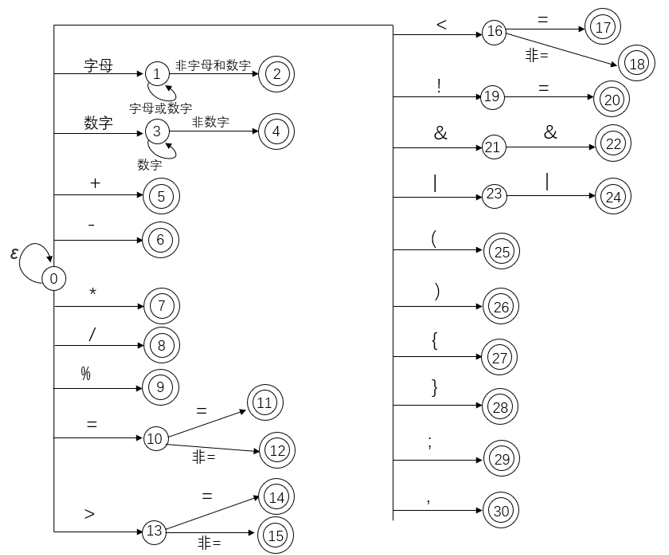
#### 3.3 扫描器



我们采用类似《程序设计语言编译原理》3.2.2节中介绍到的超前搜索方法：采用两个指示器，一个指向当前正在识别的单词的首字符，另一个向前搜索以寻找单词的终点。

如果搜索指示器没有发现更远的可行终态，返回机制会触发，快速回到前一个有效状态。

#### 3.4 NFA生成



NFA自动构建的过程是基于正则表达式的AST（Abstract Syntax Tree）解析和NFA生成。具体而言，使用Dijkstra双栈法来生成AST。将正则表达式解析为AST的形式。然后利用汤普森算法，根据AST构建NFAε-NFA。

在AST生成过程中，Dijkstra双栈法是一个常用的算法。它通过遍历正则表达式的字符，并根据操作符的优先级构建AST。这样可以保证生成的AST具有正确的结构。

接下来，利用汤普森算法将AST转换为NFAε-NFA。汤普森算法的主要思想是根据AST的节点类型执行不同的操作。例如，连接操作将两个NFA合并为一个，选择操作创建两个NFA并通过ε转换关联起来，闭包操作允许重复匹配一个NFA。通过这些操作，可以逐步构建NFAε-NFA。

#### 3.5 DFA确定化

DFA确定化的过程是基于NFAε-NFA转换为DFA的算法。从NFAε-NFA的起始状态开始，根据输入字符进行转换，获取到达的状态集合。对于新的状态集合，继续根据输入字符进行转换，直到没有新的状态集合产生为止。这样得到的DFA包含了确定的状态和转换规则。

具体来说，在DFA确定化的过程中，使用状态集合和转换规则的数据结构来表示DFA。通过遍历NFAε-NFA的状态集合，并根据输入字符获取到达的新状态集合，不断迭代直到没有新的状态集合产生。这样可以构建一个确定的状态集合和对应的转换规则。

#### 3.6 DFA最小化



DFA最小化的过程是通过等价关系划分状态集合，将等价的状态放在同一个子集中，并不断细分子集直到不能再细分为止。这样可以减少DFA的状态数目，提高词法分析器的效率。

在DFA最小化的过程中，使用并查集优化集合中元素的查找、修改和删除等操作。并查集是一种常用的数据结构，它允许在O(1)的时间复杂度内对集合以及集合中元素进行操作。通过维护等价关系的并查集，可以将等价的状态放在同一个子集中，并将不同子集中的元素区分开来。

#### 3.7 配置读入和Token匹配

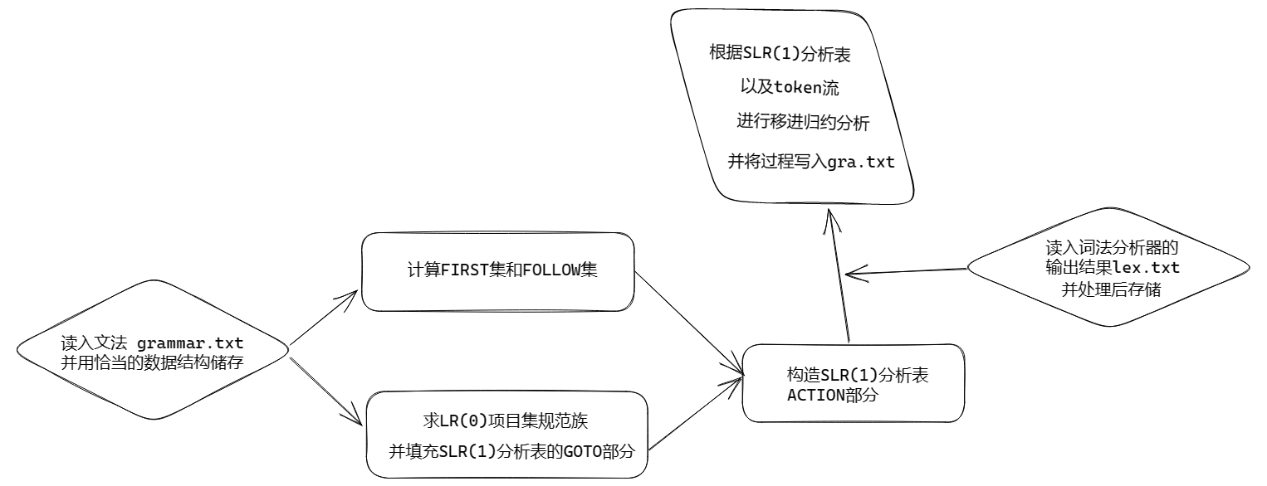
配置读入和Token匹配的过程是基于配置的转换和匹配规则。词法分析器根据输入的源代码字符，根据当前的DFA状态和转换规则，进行状态转移，并判断是否匹配到完整的词法单元。

在配置读入的过程中，词法分析器根据当前的DFA状态和输入字符，通过转换规则找到下一个状态。这样可以实现状态的转移。

在Token匹配的过程中，词法分析器判断当前的DFA状态是否为终止状态，如果是，则表示匹配到一个完整的词法单元。此时，将该词法单元输出，并将DFA状态回到起始状态，继续匹配下一个词法单元。

### 4 语法分析器描述

#### 4.1 流程图



首先读入给定的C--文法grammar.txt，并用恰当的数据结构储存，然后计算FIRST集和FOLLOW集，然后安装课本中讲授的算法计算闭包与GO函数求LR(0)项目集规范族并填充SLR(1)分析表的GOTO部分，然后构造SLR(1)分析表的ACTION部分，然后读入词法分析器的输出结果lex.txt，根据SLR(1)分析表以及token流进行移进归约分析，并将过程写入最终的输出gra.txt

#### 4.2 文法的读入处理与存储

读取文法后，使用了Python提供的多种数据结构来存储相关的信息，主要提取了它的以下关键信息：

符号信息（包括所有涉及的符号、非终结符与终结符）

原始信息（可以按行号遍历所有文法，方便后期求闭包等）

产生式头部及其产生式列表（可以以非终结符为索引获取与其相关的产生式等）

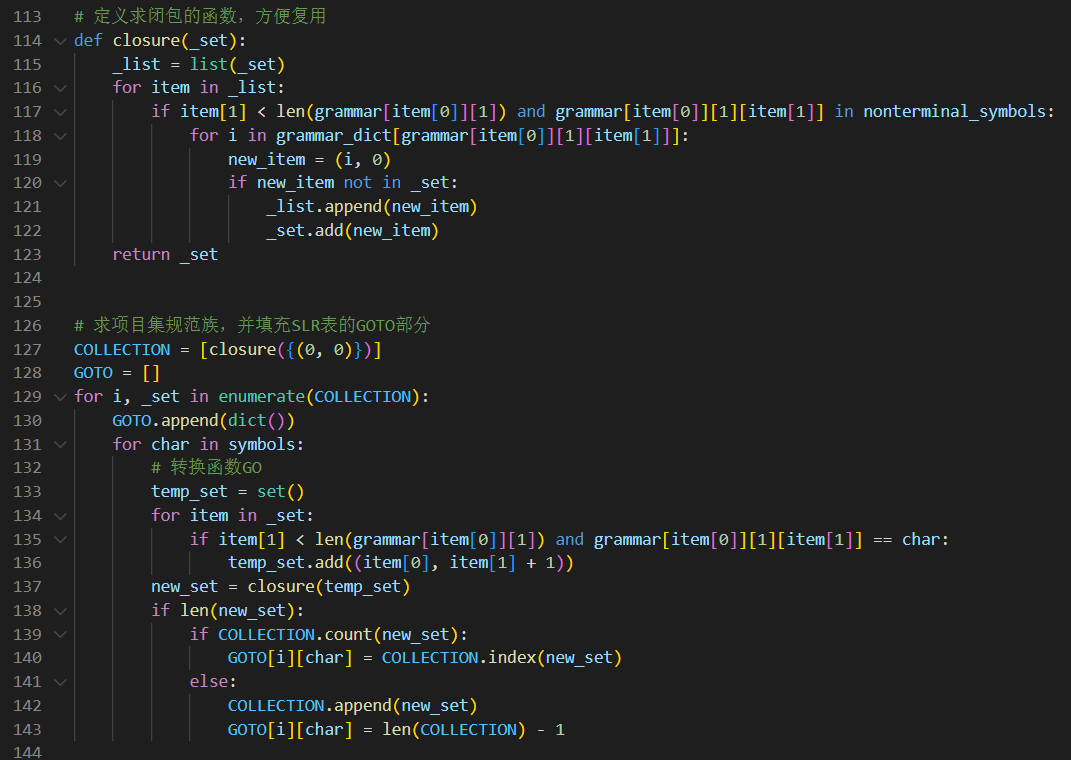


#### 4.3 FIRST集与FOLLOW集的求解

同样使用了课本中的标准方法，使用Python对其进行了实现，求解FIRST集直至FIRST集不再改变得到最终的FIRST集，FOLLOW集也使用类似的处理方法，最终达到了目的。

#### 4.4 LR(0)项目集规范族的求解

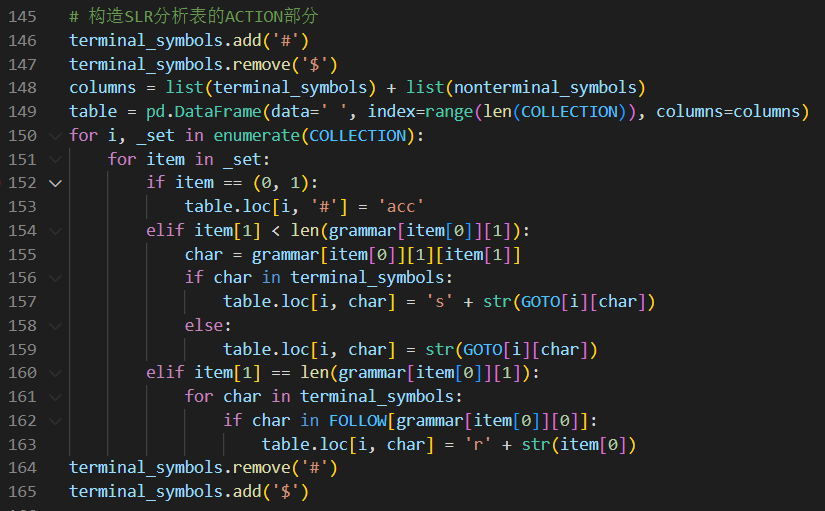
根据项目集闭包算法编写函数 closure，扩充项目集直到不再增大。根据状态转换规则使用 GO 函数: GO(I,X)=CLOSURE(J)。



#### 4.5 SLR(1)分析表的构造

根据 LR(0)项目集规范族、FIRST集和FOLLOW集构造SLR(1)分析表。分析表使用pandas的DataFrame存储。索引为状态序号，column 由除去空串并添加“#”的全体符号组成。

对于终结符，根据课本中的介绍，acc 表示接受，sj表示移进并转移到状态j。rj表示使用第j个产生式进行归约。对于非终结符，只记录转移状态j。



#### 4.6 移进归约过程

使用了三个Python中的 deque 栈来实现整个移进归约的过程。

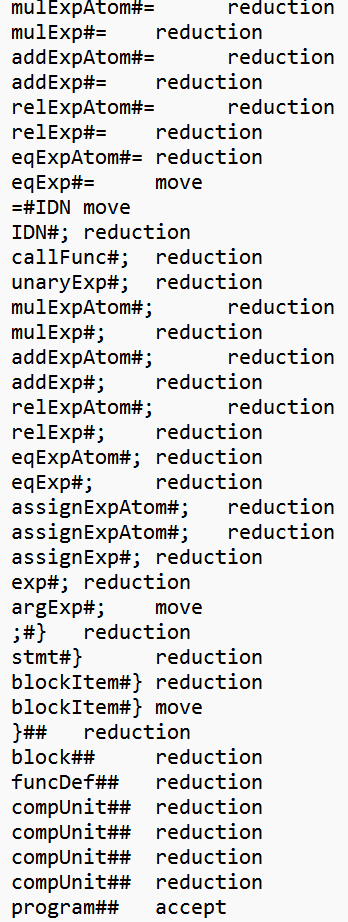
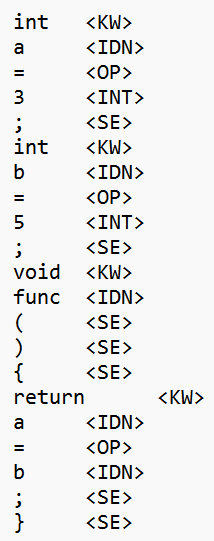
分析动作根据栈顶状态和当前输入符号确定。以此为索引访问 SLR 分析表。在SLR分析表中：对于终结符，acc 表示接受，sj 表示移进并转移到状态j。rj 表示使用第 j 个产生式进行归约。对于非终结符，只记录转移状态 j。

从而得到当前应该采取的动作。按照 LR 分析器的分析过程进行移进归约。

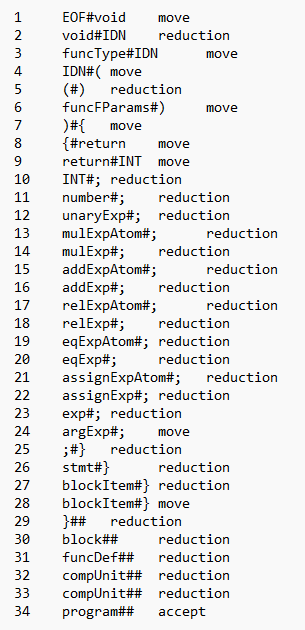
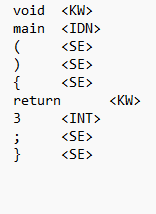


### 5 样例测试

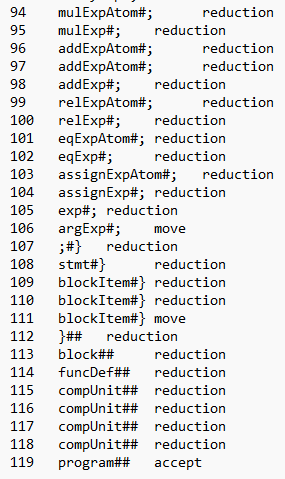
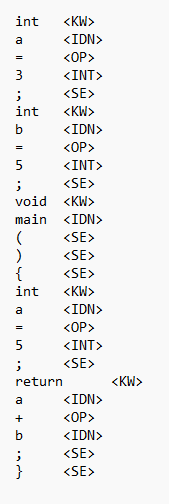
验收过程中的**test2.txt**输出结果如下：



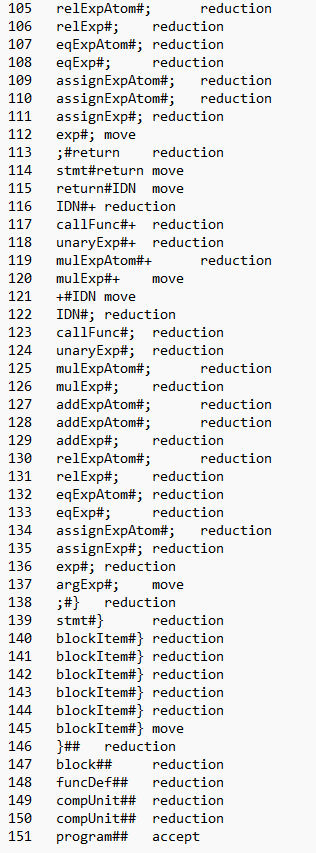
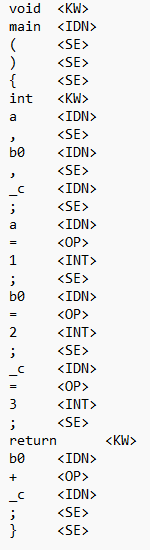
测试用例**00**输出结果如下：



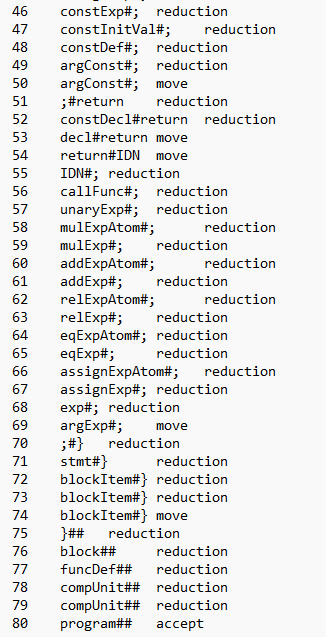
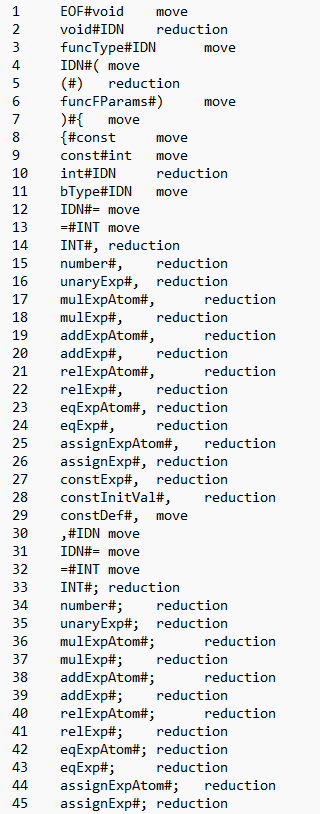
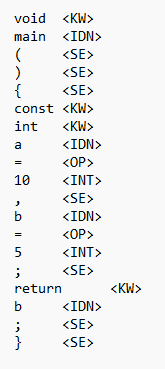
测试用例**01**输出结果如下：



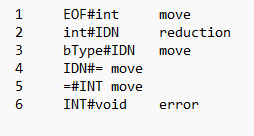
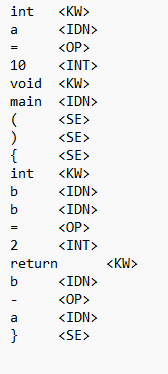
测试用例**02**输出结果如下：



测试用例**07**输出结果如下：



测试用例**10**输出结果如下：



### 6 参考资料

-《程序设计语言编译原理\_第3版》

-《现代编译原理C语言描述》

### 7 小组分工

吴显宗：词法分析器，中间代码，编写报告，ppt展示

刘宏伟：语法分析器，统筹思路，编写报告，ppt展示

吴金垚：画图，编写报告

邱胜慧：测试样例数据