**编译原理课程实验报告**

**实验2：语法分析**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 王少博 | | 院系 | 计算机科学与技术学院 | | | 学号 | | 181110315 | |
| 任课教师 | | 韩希先 | | | | 指导教师 | 韩希先 | | | |
| 实验地点 | | 研究院中507 | | | | 实验时间 | 2020.11.5 | | | |
| 实验课表现 | | 出勤、表现得分 | | |  | 实验报告  得分 |  | 实验总分 | |  |
| 操作结果得分 | | |  |
| **一、实验目的** | | | | | | | | | | |
| 基本目的：   1. **巩固对语法分析的基本功能和原理的认识。**   自底向上的语法分析的基本功能，是从给定的输入符号串出发，试图自底向上地为其建立一棵语法分析树。自底向上的语法分析的原理是从输入串出发，反复利用产生式进行归约，如果最后能得到文法的开始符号，则输入串是相应文法的一个句子，否则输入串有语法错误。在分析过程中，寻找当前句型最左的和某个产生式的右部相匹配的子串，用该产生式的左部符号代替该子串。如果每步都能正确选择子串，就可以得到输入串的最左归约过程，即规范归约过程。   1. **通过对语法分析表的自动生成加深语法分析表的认识。**   通过对闭包的求解、*first*集的求解、*follow*集的求解、*action*表的求解、*goto*表的求解，从而自动生成语法分析表。首先求I的闭包 CLOSURE(I)，然后求 I 在遇到某个文法符号 X 之后的后继项目集 GO(I, X) ,不断地循环求出 G’ 的项目集规范族 C 。接着只要求出 FIRST(X) 进而求出 FOLLOW(A) 以后，再加上项目集规范族 C 就能得到 SLR(1) 分析表了。   1. **理解并处理语法分析中的异常和错误。**   语法分析中的异常和错误主要是移进归约冲突和归约归约冲突，通过建立包含出错处理的优先矩阵来进行处理。LR 分析过程中，如果遇到异常和错误，则会中止分析，并且可以根据分析过程的输出来查看错误发生的位置、种类以及原因。  三次实验所有代码已部署于github，<https://github.com/HITWH18SE/Compiler-Principles> | | | | | | | | | | |
| **二、实验内容** | | | | | | | | | | |
| 1. 文法描述   **函数定义部分：**   * 程序 → 函数定义 * 函数定义 → 函数定义 函数定义 * 函数定义 → 变量类型 id ( ) { 函数块 } * 函数定义 → 变量类型 id ( 传入参数 ) { 函数块 } * 传入参数 → 变量类型 id * 传入参数 → 变量类型 id , 传入参数 * 函数块 → 函数块 函数块 * 函数块 → 变量类型 id ;   **赋值部分**   * 函数块 → id = 算术表达式 ;   **循环部分**   * 函数块 → while ( 布尔表达式 ) { 函数块 }   **分支部分**   * 函数块 → if ( 布尔表达式 ) { 函数块 } * 函数块 → if ( 布尔表达式 ) { 函数块 } else { 函数块 }   **变量说明部分**   * 变量类型 → int * 变量类型 → float * 算术表达式 → 算术表达式 算术运算符 算术表达式 * 算术表达式 → - 算术表达式 * 算术表达式 → ( 算术表达式 ) * 算术表达式 → id * 算术表达式 → digit   **布尔表达式部分**   * 布尔表达式 → 算术表达式 比较运算符 算术表达式 * 布尔表达式 → 布尔表达式 && 布尔表达式 * 布尔表达式 → 布尔表达式 || 布尔表达式 * 布尔表达式 → ! 布尔表达式 * 布尔表达式 → ( 布尔表达式 ) * 布尔表达式 → true | false   **其他部分**   * 比较运算符 → < | > | <= | >= | == | != * 算术运算符 → + | - | \* | /   变量说明、赋值、表达式、分支、循环都在文法中有所体现，产生式左部为了方便都用“函数块”来表示。  **(2)** 语法分析程序的总体结构及物理实现  **总体结构：**  语法分析程序由以下8个模块组成：   1. 求文法：识别以**文本形式保存**的文法，并获得非终结符，终结符，表达式以及开始符号。 2. 求项目集闭包：求解文法的拓广文法的某个LR(0)项目集的闭包。 3. 求后继项目集：求解在遇到某个文法符号之后的后继项目集。 4. 求项目集规范族：求解文法的拓广文法的项目集规范族。 5. 求FIRST集：求解文法中每个终结符或非终结符的FIRST集。 6. 求FOLLOW集：求解文法中每个终结符或非终结符的FOLLOW集。 7. 求SLR(1)分析表：根据文法G的项目集规范族以及FOLLOW集，构建SLR(1)分析表 8. 进行LR分析：根据词法分析得到的token序列以及SLR(1)分析表，完成LR分析。   **物理实现：**  首先执行函数7)，在其内部依次执行1)到6)这6个功能函数，最后执行函数8)。具体地，首先，加载语言，求解源程序的first集和follow集，初始化SLR分析表，初始化action表和goto表，然后初始化语法分析程序的状态栈stateStack和数值栈valueStack，然后读入经过词法分析的源程序，然后开始循环对源程序进行语法分析，并且打印action表，goto表，状态栈stateStack。  **(3)** 语法分析表及其数据结构和查找算法  **数据结构：**  SLR(1)语法分析表由action表与goto表两部分组成，数据结构都为python的列表list，其内部嵌套字典dict。在外部用数字下标作为索引，代表着不同的状态，而内部用终结符或非终结符的名称作为索引，其存储的字符串即代表着该状态下遇到该字符需要执行的动作。  **查找算法：**  首先把状态0和#压入栈中，将token序列装入输入缓冲区，然后根据栈顶的状态以及输入缓冲区当前第一个字符查询SLR(1)分析表的action表，如果查询到的字符串是空的，说明出错，如果字符串是acc，说明分析成功，如果字符串第一个字母是s，后面跟着数字，说明要将缓冲区当前字符以及数字对应状态压入栈中，如果字符串第一个字母是r，后面跟着数字，说明要根据数字对应的表达式归约，将表达式里的终结符、非终结符以及状态都从栈中弹出，然后再根据栈顶的状态以及输入缓冲区当前第一个字符查询SLR(1)分析表的goto表，如果字符串为空，说明出错，否则将查询到的状态压入栈中。  **(4)** 语法分析表的生成算法  算法的大致步骤如下：  **输入：文法*G*=(*V*, *T*, *P*, *S*)的拓广文法*G*';**  **输出：*G*'的*LR*(0)分析表，即*action*表和*goto*表;**  **步骤：**   * + 1. 令*I0*= CLOSURE({*S*'→.*S*})，构造*G*'的*LR*(0)项目集规范族*C*={*I0, I1, … , In*}     2. 让*Ii*对应状态*i*，*I0*对应状态0，0为初始状态。     3. **For** *k*=0 **To** *n* **Do**        - 1. **If** *A*→*α*.*aβ* ∈ **And** *a*∈*T* & GO(, *a*)= **then** *action*[*k*, *a*]=;          2. **If** *A*→*α*.*Bβ* ∈ **And** *B*∈*V* & GO(, *B*)= **then** *goto*[*k*, *B*]=*j*;          3. **If** *A*→*α*.∈**And** *A*→*α*为*G*的第*j*个产生式**then**   **For** ∀*a*∈FOLLOW(*A*) **Do**  *action*[*k*, *a*]=*rj*;  **End**   * + - * 1. **If** *S*'→*S*.∈**then** *action*[*k*, #]:=acc   **End**   * + 1. 上述⑴到⑷步未填入信息的表项均置为error。   本次实验要构建的语法分析表为SLR(1)分析表，而SLR(1)分析表是建立在LR(0)分析表基础上的，因此首先需要完成LR(0)分析表的前置条件，再加上FOLLOW集即可。  根据构建算法，对于文法G的拓广文法G’的一个LR(0)项目集I，程序首先需要求I的闭包CLOSURE(I)，然后要求I在遇到某个文法符号X之后的后继项目集GO(I, X)。在不断地循环求出所有的项目集I以后，这些项目集I就是G’的项目集规范族C。接着只要根据项目规范族C就能构建出对应的LR(0)分析表。  在LR(0)分析表的构建函数中修改其中一部分规则，就能得到SLR(1)分析表，但这部分规则需要求某个非终结符A的FOLLOW集，而求FOLLOW(A)又需要每个终结符或非终结符X的FIRST集，即FIRST(X)。所以在求出FIRST(X)进而求出FOLLOW(A)以后，再加上项目集规范族C就能得到SLR(1)分析表了。  **(5)** 错误处理  生成文法G对应的SLR(1)语法分析表时，可以通过查看项目集规范族C的构建过程以及FOLLOW集的求解过程来判断哪里出现了问题，如果文法G本身没有问题，则在LR分析以前不会出现错误。如果SLR1分析表中产生移进规约冲突，就报SLR1 Conflict error错误，然后程序退出。如果在进行LR分析时出错，说明是词法分析得到的token序列不符合语法，在读取到某个字符时，程序在SLR(1)分析表的对应位置查询到的动作是空白，说明无法继续归约下去，程序也会就此中止。因此只要查看程序输出的LR分析过程，就能查找到错误发生的位置以及类型。 | | | | | | | | | | |
| **三、实验结果** | | | | | | | | | | |
| 三次实验所有代码已部署于github，<https://github.com/HITWH18SE/Compiler-Principles>   1. 测试样例1：     图1 测试样例1  如图1所示为一个没有语法错误的正确测试样例，预计的语法分析输出结果为分析成功。  语法分析结果：（由于篇幅有限，输出结果超i过1000行，只显示部分结果）        图2 测试样例1结果  如上图所示为部分LR分析过程及其结果，可以看出输入缓冲区的字符按照SLR(1)分析表进行压栈和归约，最终分析成功，与预期输出结果一致。  （2）测试样例2：    图3 测试样例2  如图所示为一个有语法错误的测试样例，预计的输出结果为分析错误，并且显示出错时的字符位置。  语法分析结果：        图4 测试样例2结果  输入缓冲区的字符按照SLR(1)分析表进行压栈和归约，最终在id / id + id之间缺失的=符号处弹出了分析出错的提示，与预期输出结果一致。 | | | | | | | | | | |
| **四、实验中遇到的问题总结** | | | | | | | | | | |
| （二）思考题的思考与分析  思考题1：给出在生成语法分析表时所遇到的困难，以及是如何处理的？  思考题2：思考还可以什么形式来给出语法分析的结果？  思考题3：如果在语法分析中遇到了语法错误，是应该中断语法分析呢，还是应该进行适当处理后继续语法分析，你是怎么处理的？  （一）实验过程中遇到的问题以及解决方法  **问题1：FIRST集与FOLLOW集的代码实现**  在按照伪代码编写FIRST集与FOLLOW集的实际代码时，“集合不再变化时停止循环”这个描述是比较含糊的，这个时候我们需要更加形式化的判断方法来解决这个问题：增加一个标记变量，标记一下集合中的元素个数是否发生变化。  **问题2：文法G的定义**  类C语言文法大多是适用于LL(1)分析法的文法，即消除了左递归的文法，但它并不适用于自底向上分析的SLR(1)分析法。因为SLR(1)分析法不存在“试探”的过程，那些消除了左递归的表达式虽然能推出，但输入缓冲区对分析表不会进行这种试探，所以必须要另一种格式的文法才行。在便捷性和适用性的权衡中，我们把大多产生式左部设置为函数块。  （二）思考题的思考与分析  **思考题1：给出在生成语法分析表时所遇到的困难，以及是如何处理的？**   1. 搞清楚了哪些算法需要G，哪些需要G’：求C与FOLLOW集的算法有的需要文法G，有的又需要它的拓广文法G’，而生成SLR(1)的函数一开始的输入文法又是G’，很容易在调用函数时将这两个文法搞混，导致结果出错。有时需把生成SLR(1)的函数的输入文法改成了G，在函数内部再进行拓广。 2. 对token序列进行预处理，建立一个输入缓冲区是比较必要的：对token序列进行LR分析时，我发现不能直接采用token序列，因为文法G的终结符有时是大类，比如标识符id，而这是token序列里的种类，有时又是具体的属性值，比如具体的保留字、关键字。   **思考题2：思考还可以什么形式来给出语法分析的结果？**  以语法分析树的形式给出语法分析的结果，每次按照SLR(1)表指示归约时，将表达式左部的字符作为父结点，表达式右部的字符作为子节点，这样自底向上地构造出一棵树，如果分析成功，则树的根节点就是文法G的开始字符S。还可以通过绘制识别拓展文法全部活前缀的DFA，从图中看出各个状态的转换，然后构造语法分析表，通过绘制有限确定状态机来显示语法分析的结果。  我也提供了语法分析树的结果（给出样例2）：  图5 语法分析树结果  **思考题3：如果在语法分析中遇到了语法错误，是应该中断语法分析呢，还是应该进行适当处理后继续语法分析，你是怎么处理的？**   1. 如果是产生了移进规约冲突，或者是规约规约冲突，我们不必中断语法分析，可以对程序采取紧急方式恢复策略，发现错误时跳过一些输入符号，直到出现下一个语法成分包含的第一个符号为止，在实现的时候，通常用某个期望的同步记号作为相应的标记，这种错误处理方法的效果依赖于属于follow集合的同步记号的选择。 2. 如果是其他未知的语法错误，事先没有准备的，就可以中断语法分析，因为通常在这种情况下是很难处理的，而且这也是很少出现的。在语法分析过程中遇到了这类错误应该中断语法分析，也就意味着当前栈顶状态与输入缓冲区最前面的字符在SLR(1)分析表中对应的动作字符串是空白，接下来无论如何不可能继续归约下去。 | | | | | | | | | | |
| **五、实验体会** | | | | | | | | | | |
| 通过语法分析的实验，对LR(0)分析法和SLR(1)分析法都有了更加深刻的了解。 LR(0)分析法不需要向前查看输入符号，只需要根据当前的栈顶状态就可以确定下一步所应采取的动作。但是，SLR(1)分析法让分析器向前查看一个输入符号，以便确定面对当前输入符号时是否进行规约，以及按照哪个产生式进行规约。只需要在LR(0)分析法的算法中更改一小部分代码即可。理解构造SLR(1)分析表的过程以后，语法分析也没有太多的难度，这些函数只要在合适的地方调用，就可以很顺利地完成整个流程。模块化编程的作用确实很大，划分功能子模块之后编程如鱼得水。  实现算法的顺序和书本的教学顺序是一样的，先实现闭包CLOSURE，然后实现后继GO，然后实现项目集规范族C，然后实现LR(0)分析表的构造，然后实现求FIRST集与FOLLOW集，最后修改LR(0)分析表的代码，实现SLR(1)分析表的构造。  在实现这些函数的过程中，难度最高的两个算法还是求FIRST集与FOLLOW集，书上的伪代码算法描述比较抽象，好在python语言实现比较简洁，最后还是解决了。同时，文法G的定义是关键，不能只考虑LR(0)分析法的情况，还需考虑SLR(1)分析法。  总之，在此次实验中，获益匪浅，独立完成实验的体验成就感很大。 | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | |