语法分析实验报告

------2020200982 闫世杰

实现简单的语法分析器,并生成语法树
实现思路:将根据词法分析得到的词组作为终结符传递给语法分析器,利用语法分析器进行语法分析,在进行语法分析的过程中,把得到的结果储存在树形结构中,最后 DFS 遍历树形结构,根据树形结构的父子关系建立语法树图片的基本框架结构,根据节点中储存的值确定框架中各个节点块中的内容,生成所需的 dot 文件,利用 graphviz 转化为图片即可

树形结构:

```
struct Node{
    char* type; //记录节点的类型信息,可以是非终结状态名称,终结符名称等
    char* id; //如果节点是标识符,记录标识符的名称,输出 dot 文件时使用
    int number; //如果节点是数字,记录数字的值,输出 dot 文件时使用
    struct Node**child; //指向孩子节点
    struct Node* p; //指向父亲节点
    int c_n; //记录孩子的个数
    int nth; //记录当前节点的编号,输出父子关系 nodex:fx->nodey 时使用
}
```

树形结构的实现:
创建节点,由于创建标识符节点和数字节点时需要额外的数据支持,使用 3 个不同的函数
Node* Make1(char*type):创建普通节点,直接将设置 type,初始化各项数据即可
Node* Make2(int n):创建数字节点,type=Number,number=n,初始化各项数据
Node* Make3(char*id):创建标识符节点,type=id,id=id,初始化各项数据
void Insert(Node*p, Node*c):将 c 插入到 p 的孩子中,c 的父亲指针设为 p;
int IfKey(Node*p, int th):
生成语法树图片用到 graphviz,受其语法规则的限制,部分符号需要进行转义才能输出出来,包括{ } > <,因此一些终结符需要特殊处理输出:{ } > < >= <=,同时 number 和 id 类型的节点在输出是需要输出具体的内容,因此将上述的节点的输出单独拿出来进行特殊处理
th 代表 p 是 p->p 的第 th 个孩子,输出关系指向 <fth> 时需要用到

void DFS(Node*p, int n):
对树形结构进行递归遍历,根据树形结构中节点的父子关系以及储存的内容确定语法树的基本框架结构以及文本框中的内容,并根据graphviz语法规则将具体的语句输出到dot文件中
void Create(Node*p):
进行一些初始化输出,包括文件的打开及初始化,然后调用 DFS 进行 dot 文件的生成

在词法分析中,直接将 yytext 作为 type 参数传入来建立节点
在语法分析中,直接将状态的名称作为 type 参数传入来建立节点
因此对于大部分节点,输出文本框中具体内容时比较简单,只需将 type 中的内容包装起来即可,但是对于一些需转义的内容以及标识符和数字,注意特殊处理
语法基本框架的输出以及将文本内容包装成具体的 graphviz 语句比较繁琐,但由于是递归遍历树形结构实现,只需根据具体例子生成一个基本情况,然后递归调用即可

词法分析:
词法分析的任务是匹配关键字(含界符) 数字 标识符,并将其作为终结符传递给词法分析器
对于 3 中不同的情况
1.关键字:调用 Make1(yytext),将关键字作为 type 建立节点,同时 return 对应的终结符
2.数字: 将 yytext 匹配到的数字语句转化成数字 number(需区分 8 10 16 进制),调用 Make2(number)生成数字节点,同时 return 终结符 Number 给语法分析器
3.标识符:调用 Make3(yytext),生成标识符节点,同时 return 终结符 ID 给语法分析器

在词法分析的过程中也需要进行节点的创建,这些终结符节点最终都会在树形结构中作为叶子节点出现,同时还需要将匹配到的内容所对应的终结符返回给语法分析器,以便语法分析进行移进和归约
由于每个关键字都对应着不同的终结符,不同进制数字转化成 int 类型的过程不尽相同,因此,基本每个匹配都需要根据具体的 yytext 才能确定所需要返回的终结符,情况比较多
由于没有实现错误恢复,识别错误时只是简单的输出 error,因此行列号没有进行使用,可以在树形结构叶子节点中添加所在行列号的信息,以便进行错误位置的输出(未具体实现

语法分析:
由于有现成的语法分析器生成工具,因此只需要写出具体的语法规则即可
具体的语法规则在文档中都已经给出了,只需要稍加改进,实现其中包含的正则匹配式,同时定义相关的优先级或者改写文法来消除冲突即可
正则匹配式的实现:
1.[T]:T 可有可无的情况
    1)可以改写成 T' -> T|empty 形式
    2)也可以直接把包含[T]的语句写成两句,一个含 T 一个不含 T(本人采用)
2.{T}:T 可以 0 次或多次出现
    利用左递归,写成 Ts->Ts T|empty 即可
冲突的消除:
1.四则运算优先级问题
在文档给出的语法规则中加减语句定义如下
```
AddExp:  MulExp
        |AddExp +/- MulExp
```
在这种定义下,文法自动地先分析乘除再分析加减,因此四则运算优先级问题可以顺利解决
2.if else 移进归约问题
使用%prec 和%nonassoc 配合进行优先级的定义,在 if 语句后加上%perc xxx 使得 if else 的优先级高于 if 即可解决
文法修改完成之后,只需要在相应的语法规则之后执行相应的动作即可,包括 detail 的输出以及各个状态的树形节点的创建以及树形结构中父子节点关系的创建

make 之后
执行./myan1 < test > detail.txt
由于 detail 使用 printf 输出,dot 文件使用 fprintf 指定了输出文件,main 函数较为简陋
因此使用重定向进行输入,再把结果重定向输出到 detail.txt 文件中
在执行 myan1 过程中会自动创建和完成 Tree.dot 文件
最后执行 dot -Tpng Tree.dot -o pngname 即可得到 png

由于细节的实现不同,语法分析的状态个数及名称也会有区别,
由于没有实现错误恢复,因此词法和语法分析的部分比较容易实现,难点在于如何将语法分析得到的结果保存下来并进行可视化,
词法分析部分感觉像是在进行精确匹配(实际用到正则匹配的只有标识符和数字),
语法分析部分,由于绝大多数语法规则都已经给出,需要修改的地方在实验指南中都已经给出,只需要照着修改就行
最开始的想法是直接在进行语法分析的时候把匹配结果包装成graphviz语句生成dot文件,但是在执行过程中发现这样做虽然很容易就能输出文本框中的内容,但是由于很难确定父子关系的指向,因此语法树的基本框架很难建立,最后选择将匹配的结果暂存在树形结构中,最后遍历树形结构来生成dot文件