编译原理实验-PL0自底向上语法分析

原创

zekdot 于 2019-11-29 19:04:57 发布 1930 收藏 10

文章标签： 编译原理 PL0 自底向上分析

本文链接：https://blog.csdn.net/zekdot/article/details/103315733

最近顶着考研的压力去做了一下编译原理的实验，实验的要求是写一个PL/0语法的编译器，一开始想从网上找找有没有现成的代码改一改就完事了，结果百度的结果都是递归下降分析，而老师的课程大部分都在讲自底向上分析的知识，而我甚至不知道这个文法是不是SLR的，所以为了讲实验一咬牙干脆自己写一个好了，如果算出来是SLR的就去讲实验，不是SLR大不了分不要了，因为LL(1)表的构造实在是让人难受，幸好结果证明这文法确实是SLR的。

有限的时间中大多数都花在了构造语法分析表的部分，所以后面的翻译执行并没有完成。

完整的python实现程序已经上传到了码云，需要的话可以参考：

编译原理实验-PL0自底向上分析

1.原文法

〈程序〉→〈分程序>.

〈分程序〉→[<常量说明部分>][<变量说明部分>][<过程说明部分>]〈语句〉

<常量说明部分>→CONST<常量定义>{ ,<常量定义>};

<常量定义>→<标识符>=<无符号整数>

<无符号整数>→<数字>{<数字>}

<变量说明部分>→VAR<标识符>{ ,<标识符>};

<标识符>→<字母>{<字母>|<数字>}

<过和说明部分>→<过程首部><分程度>；{<过程说明部分>}

<过程首部>→procedure<标识符>;

<语句>→<赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语句>|<写语句>|<复合语句>|<空>

<赋值语句>→<标识符>:=<表达式>

<复合语句>→begin<语句>{ ； <语句>}end

<条件>→<表达式><关系运算符><表达式>|ood<表达式>

<表达式>→[+|-]<项>{<加减运算符><项>}

<项>→<因子>{<乘除运算符><因子>}

<因子>→<标识符>|<无符号整数>|(<表达式>)

<加减运符>→+|-

<乘除运算符>→*|/

<关系运算符> → =|#|<|<=|>|>=

<条件语句> → if<条件>then<语句>

<过程调用语句> → call<标识符>

<当型循环语句> → while<条件>do<语句>

<读语句> → read(<标识符>{，<标识符>})

<写语句> → write(<标识符>{，<标识符>})

<字母> → a|b|c…x|y|z

<数字> → 0|1|2…7|8|9

2.词法分析

这里我用字符串分析的方法来做的，具体逻辑可以见代码语法分析树文件夹下的tokens.py。在词法分析的同时，我把一部分非终结符变成了终结符方便处理：

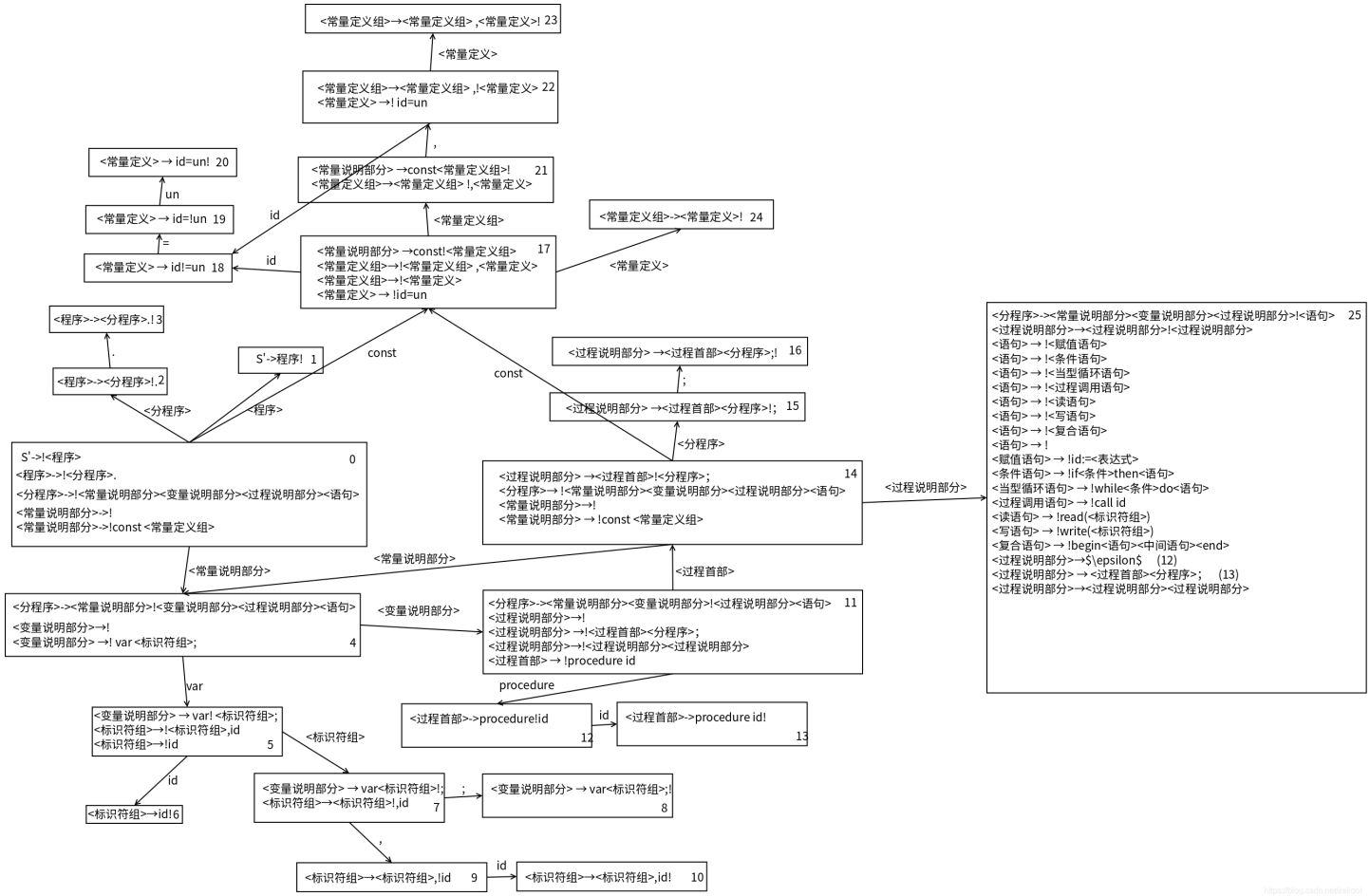
```
终结符
const var procedure begin end ood if then while do read call write
 := = { } ( ) ; , .
id-<标识符>
un-<无符号整数>
re-<关系运算符>   #|<|<=|>|>=
mn-<加减运算符>  +|-
td-<乘除运算符>  *|/
```

3.状态分析表构造

自底向上的分析方法需要状态分析表才能够实现，所以其实我最主要干的事就是造了这个分析表。
首先需要改写文法，因为当前这种文法表示形式是不能去用所学的算法构造分析表的，所以我改成了等价文法(其实并不等价，那个#当做终结符用了，所以其实最后结果中的#并没有当不等号用，如果需要可以再完善这个文法)：

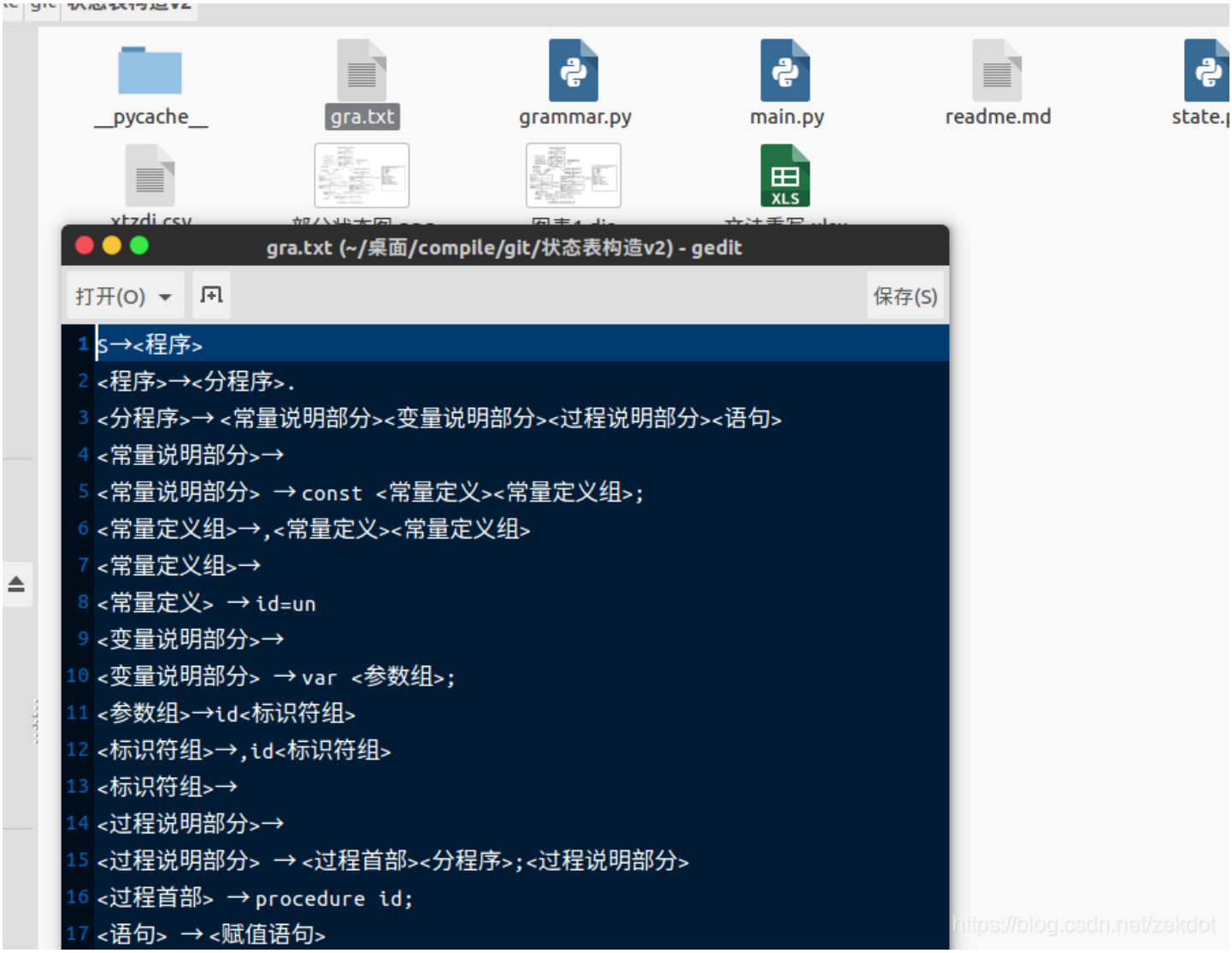
S→<程序>
<程序>→<分程序>.
<分程序>→ <常量说明部分><变量说明部分><过程说明部分><语句>
<常量说明部分>→
<常量说明部分> → const <常量定义><常量定义组>;
<常量定义组>→,<常量定义><常量定义组>
<常量定义组>→
<常量定义> → id=un
<变量说明部分>→
<变量说明部分> → var <参数组>;
<参数组>→id<标识符组>
<标识符组>→,id<标识符组>
<标识符组>→
<过程说明部分>→
<过程说明部分> → <过程首部><分程序>;<过程说明部分>
<过程首部> → procedure id;
<语句> → <赋值语句>
<语句> → <条件语句>
<语句> → <当型循环语句>
<语句> → <过程调用语句>
<语句> → <读语句>
<语句> → <写语句>
<语句> → <复合语句>
<语句> →
<赋值语句> → id:=<表达式>
<条件语句> → if<条件>then<语句>
<条件> → <表达式>re<表达式>
<条件> → ood<表达式>
<表达式> → <项><表达式组>
<表达式组> → <加减运符><项><表达式组>
<表达式组> →
<加减运符>→mn
<项>→<因子><因子组>
<因子组> → td<因子><因子组>
<因子组>→
<因子> → id
<因子> → un
<因子> →(<表达式>)
<读语句> → read(<参数组>)
<写语句> → write(<参数组>)
<复合语句> → begin<语句><中间语句>end
<中间语句>→;<语句><中间语句>
<中间语句>→
<当型循环语句> → while<条件>do<语句>
<过程调用语句> → call id

一开始手算了一点，算着算着就发现这玩意手算估计我当天晚上就不用睡觉了：



于是决定开始写程序去算，具体的逻辑实在太过复杂，First，Follow还有最后的状态收集打印都花了我不少时间，这里直接讲述一下分析表生成程序的用法，所有关于分析表生成的程序都在状态表构造V2目录下：

3.1.将文法保存在gra.txt



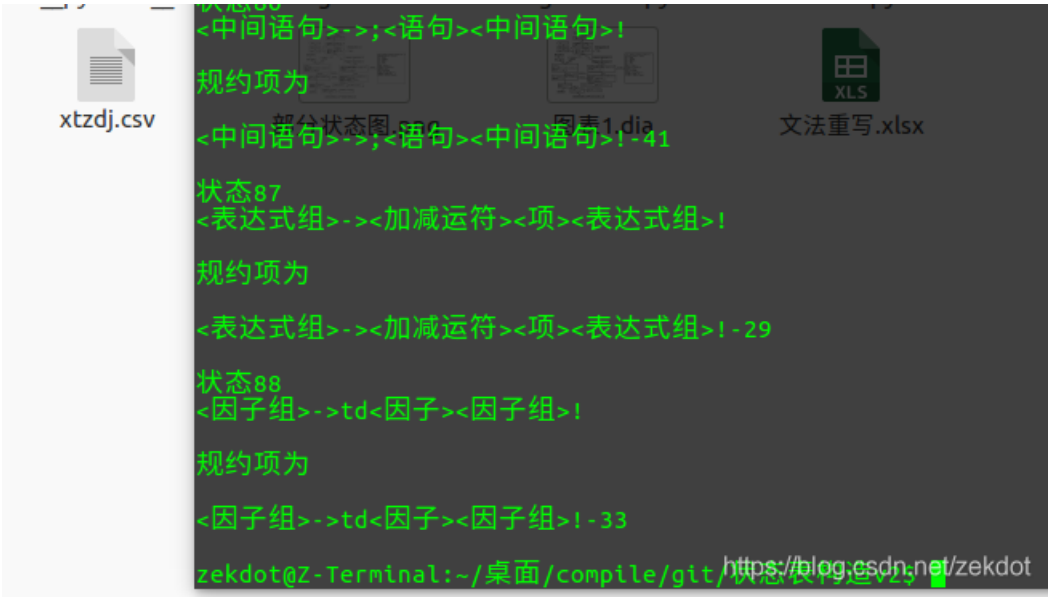
如图，将自己的文法写到gra.txt中。

3.2.运行程序得到分析表

在命令行键入

```
python3 main.py
```

即可运行程序



如图，程序运行会打印出所有状态，可以利用管道把这个状态输出到文件中便于进行调试。这里的xtzdj.csv就是生成的分析表，可用Excel打开查看：

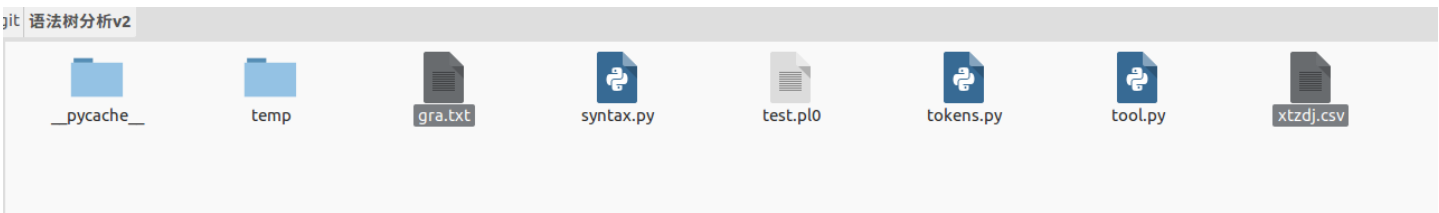
		const	var	procedure	begin	end	ood	if	then	while	do	read	call	write	#	:=	=	(	)	;	逗号	.	id	un	re
42	40																			r7	r7				
43	41	r15	r15	r15	r15			r15		r15		r15	r15	r15						r15			r15		
44	42										s59														
45	43					r35			r35		r35								r35	r35		r35			r35
46	44																	s44					s43	s49	
47	45					r30			r30		r30								r30	r30		r30			r30
48	46																								s64
49	47																	s44					s43	s49	
50	48					r34			r34		r34								r34	r34		r34			r34
51	49					r36			r36		r36								r36	r36		r36			r36
52	50																	s44					s43	s49	
53	51					r42														s70					
54	52					r44														r44		r44			
55	53								s71																
56	54																						s14		
57	55																						s14		
58	56			s10	r13			r13		r13		r13	r13	r13						r13		r13	r13		
59	57																		r12	r12	s36				
60	58																			r5					
61	59				s25	r23		s27		s20		s29	s26	s33						r23		r23	s22		
62	60																		s77						
63	61																	s44					s43	s49	
64	62																	r31					r31	r31	
65	63					r28			r28		r28								r28	r28		r28			r28
66	64																	s44					s43	s49	

这里行是状态，列是终结符或者非终结符，表中单元则是转移或者规约动作。

4.进行语法分析

得到语法分析表之后，就可以愉快的分析源程序的语法了，其实编写程序的过程并不愉快，文法的读取、状态的收集去重等还是花了我很长时间，这里也不具体讲述代码了，简单说一下程序执行方式：

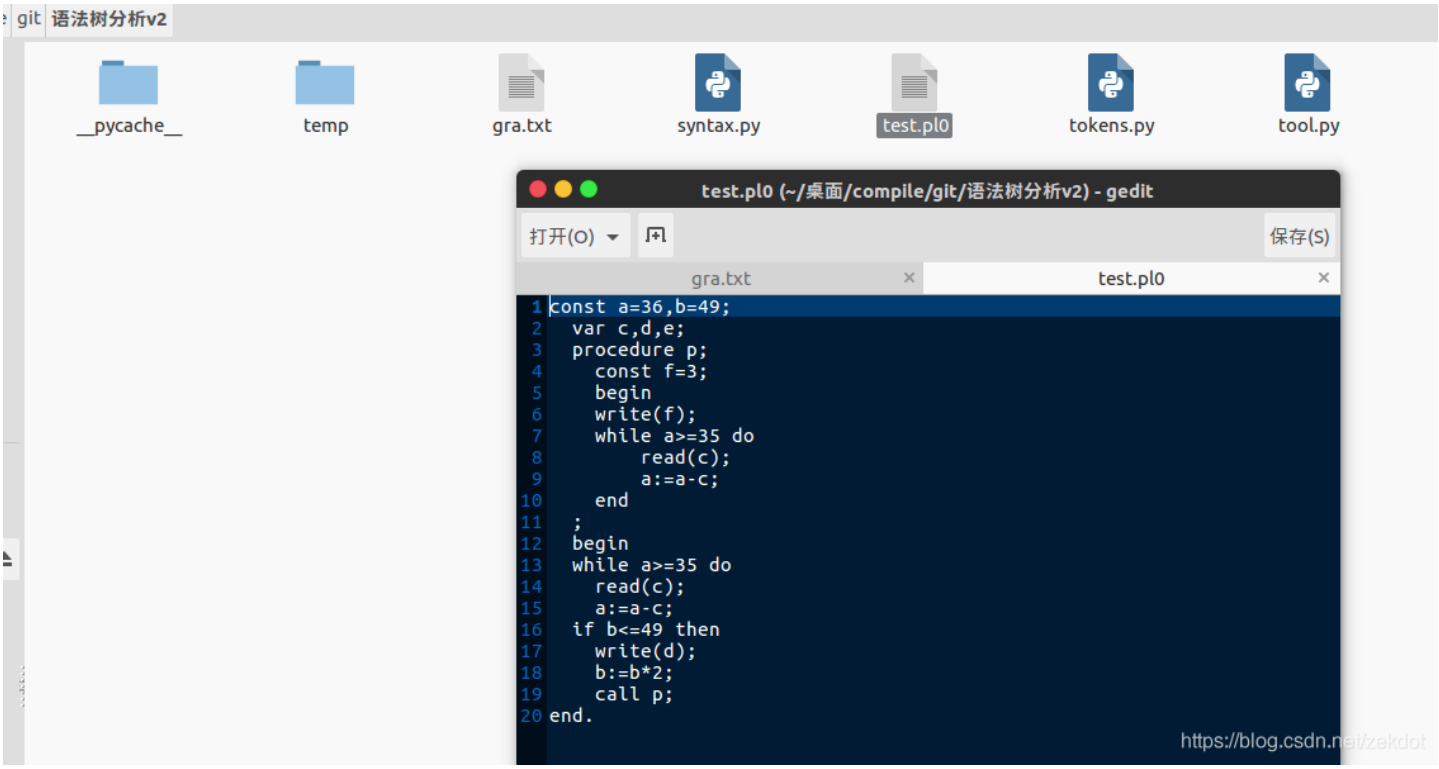
4.1.把文法和语法分析表拷贝到语法分析树v2目录下



如图，把这两个文件放到该目录下。

4.2.把要分析的源码写到目录下的test.pl0中

这里我的逻辑默认从test.pl0中读取源代码分析，如果需要的话可以修改syntax.py来修改这个默认名称。

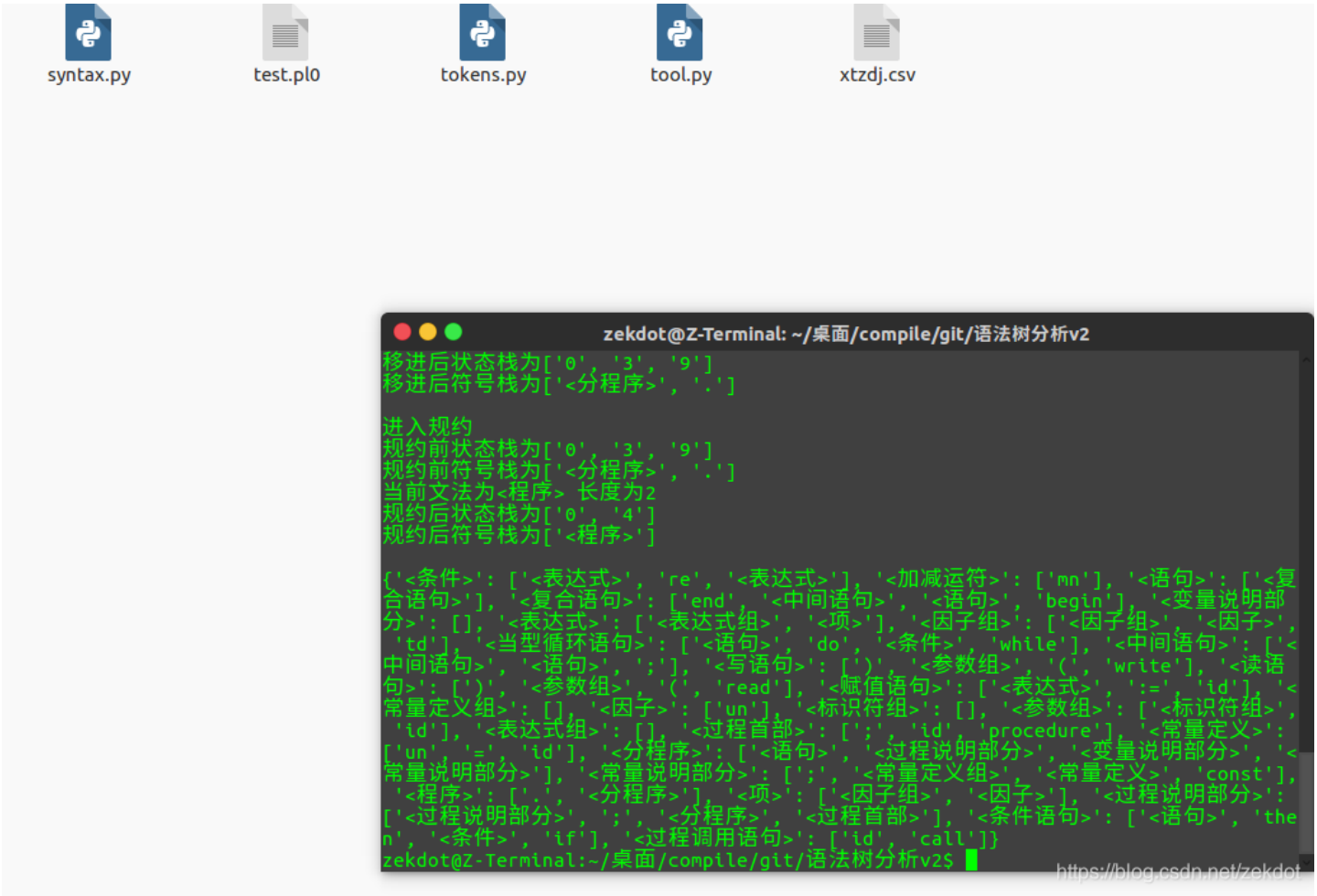


4.3.运行语法分析程序

在命令行中键入

```
python3 syntax.py
```

即可运行程序。



如图所示，成功的识别了测试的pl0程序，并打印出了语法树的树枝。即语法分析成功。

其实到这一步整个编译器的工作并没有完成，接下来应该为其编写对应的属性文法，然后把源代码翻译成四元组，然后再是代码的优化，最后是执行，剩下的部分如果以后能闲下来再去探究吧。