

Syntax Parser Programming实验报告

71114223 陈雄辉



2017-1-14

东南大学

计算机科学与技术和软件学院

目录

[1 Motivation/Aim 2](#_Toc472189119)

[2 Content description 2](#_Toc472189120)

[3 Ideas/Methods 2](#_Toc472189121)

[3.1 基础算法，求First 2](#_Toc472189122)

[3.2 构造预测分析表 2](#_Toc472189123)

[3.3 根据预测分析表生成的字典，进行token序列的分析 3](#_Toc472189124)

[4 Assumptions 3](#_Toc472189125)

[4.1 实验环境： 3](#_Toc472189126)

[4.2 对input文件的规定： 3](#_Toc472189127)

[5 Related FA descriptions 4](#_Toc472189128)

[6 Description of important Data Structures 7](#_Toc472189129)

[7 Description of core Algorithms 8](#_Toc472189130)

[7.1 构造分析表的程序分析 8](#_Toc472189131)

[7.2 状态内扩展和状态间扩展 9](#_Toc472189132)

[7.3 规约和移进 12](#_Toc472189133)

[7.4 根据预测分析表进行语法分析： 13](#_Toc472189134)

[8 Use cases on running 15](#_Toc472189135)

[8.1 输入文件： 15](#_Toc472189136)

[8.2 输出文件： 16](#_Toc472189137)

[9 Problems occurred and related solutions 20](#_Toc472189138)

[10 Your feelings and comments 21](#_Toc472189139)

# Motivation/Aim

本试验为了实现LR（1）文法分析器，实现以的文法产生式和tokens序列为输入，能够自动生成确定化状态图，预测分析表，以及分析队列

本项目源码已经上传到gihub，地址：

<https://github.com/Bunrning-Bear/compiler_experiment_python>

# Content description

本试验要做以下几个工作：

* 根据输入的产生是字符串构造对应的数据结构存储
* 根据输入的token序列构造对应的数据结构存储
* 通过内部状态拓展和状态间拓展，生成对应的状态集
* 根据状态集构造预测分析表
* 根据预测分析表分析tokens序列，构造程序

# Ideas/Methods

### 基础算法，求First

以下是产生first的伪代码，其中production是产生式序列

if production[0] in terminal:

return production[0]

else:

for char in production:

for x in inner\_status\_dict[char]['status\_list']

first = First(production\_list[x][1])

result.append(first)

if eps not in first:

break;

## 构造预测分析表

* + - 化简产生式，将其中带有｜ 或号的分割开，
    - 状态初始化
    - 从0状态开始，将产生式加入待拓展队列中
    - 从队列中拿取一个元素进行状态内拓展
    - 拓展完毕之后对该元素进行状态间拓展，将拓展生成的新的状态(还没完成状态内拓展)加入待拓展队列
    - 在状态内拓展的过程中将规约项填入预测分析表
    - 在状态间拓展的过程中将移进项填入预测分析表
    - 重复iv到vi的动作，直到待拓展队列没有新的状态

## 根据预测分析表生成的字典，进行token序列的分析

# Assumptions

## 实验环境：

Python2.7

Ubuntu14.04

要求安装prettytable库，用于格式化输出表格

## 对input文件的规定：

* 输入文件有两个，一个是产生式序列，一个是token序列
* 对于产生式序列：
  + 每个产生式有两行，第一个是产生式信息，第二个需要注明每个产生式的符号是终结符还是非终结符，用T代表终结符，N代表非终结符，如：
    - E->E+T
    - N->NTN
  + 在我们的实验中，用户的输入的产生式的数据现在只能是分割后的产生式，也就是说，产生式右边文件不要出现|的符号
  + 第一个产生式为零号产生式，也就是说，第一个产生式应该是以下的形式：
    - S->E
    - N->N
  + 用'e'代表epsilon
  + 由于代码结构原因，目前默认每行最后一个元素是’\n’，所以，暂时需要在产生式输入文件的最后一行多一个空行”\n”
* 对于token序列：
  + 需要注明的是，我们的输入仅仅是和产生式对应的token序列，而不是真正的程序源代码

# Related FA descriptions

项目的有限自动机的每个状态存储在ParsingTableProcessor类的definited\_status\_list中, 而自动机的每个状态间的关系在构造过程直接映射到映射分析表中，也就是ParsingTableProcessor. predict\_parsing\_table中，没有直接进行存储。

以下展示demo程序中的definited\_status\_list信息：

* Demo中以以下为产生式：

S->E

N->N

E->E+T

N->NTN

E->T

N->N

T->T\*F

N->NTN

T->F

N->N

F->(E)

N->TNT

F->i

N->T

* 这是对应的构造完成的状态集：

----------------------------------

| status I0: |

| S->.E,set(['$']) |

| E->.E+T,set(['+', '$']) |

| E->.T,set(['+', '$']) |

| T->.T\*F,set(['+', '\*', '$']) |

| T->.F,set(['+', '\*', '$']) |

| F->.(E),set(['$', '+', '\*']) |

| F->.i,set(['$', '+', '\*']) |

----------------------------------

----------------------------------

| status I1: |

| F->i,set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I2: |

| F->(.E),set(['+', '\*', '$']) |

| E->.E+T,set([')', '+']) |

| E->.T,set([')', '+']) |

| T->.T\*F,set([')', '+', '\*']) |

| T->.F,set([')', '+', '\*']) |

| F->.(E),set([')', '+', '\*']) |

| F->.i,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I3: |

| S->E,set(['$']) |

| E->E.+T,set(['+', '$']) |

----------------------------------

----------------------------------

| status I4: |

| E->T,set(['+', '$']) |

| T->T.\*F,set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I5: |

| T->F,set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I6: |

| F->i,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I7: |

| F->(.E),set([')', '+', '\*']) |

| E->.E+T,set([')', '+']) |

| E->.T,set([')', '+']) |

| T->.T\*F,set([')', '+', '\*']) |

| T->.F,set([')', '+', '\*']) |

| F->.(E),set([')', '+', '\*']) |

| F->.i,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I8: |

| F->(E.),set(['+', '\*', '$']) |

| E->E.+T,set([')', '+']) |

----------------------------------

----------------------------------

| status I9: |

| E->T,set([')', '+']) |

| T->T.\*F,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I10: |

| T->F,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I11: |

| E->E+.T,set(['+', '$']) |

| T->.T\*F,set(['+', '\*', '$']) |

| T->.F,set(['+', '\*', '$']) |

| F->.(E),set(['$', '+', '\*']) |

| F->.i,set(['$', '+', '\*']) |

----------------------------------

----------------------------------

| status I12: |

| T->T\*.F,set(['+', '\*', '$']) |

| F->.(E),set(['$', '+', '\*']) |

| F->.i,set(['$', '+', '\*']) |

----------------------------------

----------------------------------

| status I13: |

| F->(E.),set([')', '+', '\*']) |

| E->E.+T,set([')', '+']) |

----------------------------------

----------------------------------

| status I14: |

| F->(E),set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I15: |

| E->E+.T,set([')', '+']) |

| T->.T\*F,set([')', '+', '\*']) |

| T->.F,set([')', '+', '\*']) |

| F->.(E),set([')', '+', '\*']) |

| F->.i,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I16: |

| T->T\*.F,set([')', '+', '\*']) |

| F->.(E),set([')', '+', '\*']) |

| F->.i,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I17: |

| E->E+T,set(['+', '$']) |

| T->T.\*F,set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I18: |

| T->T\*F,set(['+', '\*', '$']) |

----------------------------------

----------------------------------

| status I19: |

| F->(E),set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I20: |

| E->E+T,set([')', '+']) |

| T->T.\*F,set([')', '+', '\*']) |

----------------------------------

----------------------------------

| status I21: |

| T->T\*F,set([')', '+', '\*']) |

----------------------------------

# Description of important Data Structures

* 项目数据结构存储于以下几个文件中：

\main\Global.py,

\main\Production.py

\main\ParsingTable.py,

下面分别介绍

* \main\Global.py 存储项目中的全局变量，目前只存储了EPSILON = 'e'，这个变量
* \main\Production.py用于将用户定义的产生式信息转化为字符串存储下来
  + Class Token 用于描述文法符号，也就是一个token，包括了token的终结符与否和字符信息
  + Class ProItem 用于描述单个产生式，依赖于Token
  + Class ProductionSet 用于描述用户定义的所有文法产生式的集合
* \main\ParsingTable.py 用于存储构造预测分析表相关的数据结构，包括带dot和预测符的产生式，状态集合：
  + Class LR\_produc\_item：用于描述一个LR(1)分析过程中每个产生式的状态，它包含了产生式和产生式当前的dot和预测符信息
  + Class Status：记录了一个确定化的状态集合的信息，依赖于LR\_produc\_item

# Description of core Algorithms

## 构造分析表的程序分析

* 首先，我们从主程序开始，以下是main.py的主要代码，展示的是预测分析表的构造驱动程序，他有以下几个部分：
  + 初始化第一个状态processor.init\_status()
  + 用于存储待状态拓展的队列：to\_extent\_queue
  + 状态内拓展，同时将移进项填入分析表：definited\_status = processor.closure(value)
  + 状态间拓展：processor.goto(definited\_status)
  + 对拓展完毕的状态进行规约，将可规约项填入分析表：processor.regression(definited\_status, new\_status\_id)
  + 重复循环知道无法找到新的队列

def parsing\_table\_driver(processor):

"""LR(1)预测分析表构造处理器的驱动，调用它可以构造成分析表

Args:

processor: 输入是分析表处理器：

Returns:

processor，构造完成之后的构造器ParsingTableProcessor

"""

definited\_status = processor.init\_status()

# 状态间拓展使用的是广度优先遍历，这里构造了一个队列，

# 用来存放新产生的确定化状态的标号，以便接下来进行的状态内部拓展

to\_extent\_queue = Queue.Queue(maxsize=-1)

to\_extent\_queue.put([0, definited\_status])

while not to\_extent\_queue.empty():

item = to\_extent\_queue.get()

definited\_status\_id = item[0]

definited\_status = item[1]

# 进行状态间拓展

status\_dict = processor.goto(definited\_status)

for key,value in status\_dict.items():

# 状态内扩展

definited\_status = processor.closure(value)

# 确定一个状态，加入字典

old\_status\_id = processor.same\_status(definited\_status)

# print "----print definited status [id]:%s, [edge]: %s------,is old?:%s"%(definited\_status\_id,key,old\_status\_id)

# definited\_status.print\_status()

if old\_status\_id == -1:

new\_status\_id = processor.get\_next\_status\_id()

processor.add\_to\_definited\_status(definited\_status)

# 确定状态为新状态后，进行状态内的可规约符号进行规约

processor.regression(definited\_status, new\_status\_id)

# 将确定化的状态加入待拓展队列

to\_extent\_queue.put([new\_status\_id , definited\_status])

processor.add\_to\_parsing\_table('s'+str(new\_status\_id), definited\_status\_id, key)

else:

processor.add\_to\_parsing\_table('s'+str(old\_status\_id), definited\_status\_id, key)

return processor

## 状态内扩展和状态间扩展

下面我们进入processor.closure(value) 和 processor.goto(value) 两个函数，具体查看如何进行状态内拓展，我们做的主要是以下的事情：

* + - 遍历该状态的所有产生式
    - 判断产生式的状态，如果dot还没到达末尾，并且下一个符号不是终结符，才能进行状态拓展
    - 内部状态拓展：通过First(βα)计算每个新的产生式和预测符

def closure(self, Status):

"""内部状态拓展，构造闭包的算法

Args:

Status：用来子状态拓展的状态

"""

stat\_list = Status.get\_status\_list()

old\_length = 0

new\_length = Status.length

# 循环直到状态内找不到新的产生式为止,也就是两次循环的Status的产生式的个数相等

while(old\_length != new\_length):

old\_length = new\_length

for item in stat\_list:

# item 是一个lr\_produc产生式

pro\_id = item.produc\_id

# pro\_item 是一个LR产生式ProItem对象

pro\_item = self.production\_list.get\_right\_by\_index(pro\_id)

if item.dot\_location != pro\_item.length:

next\_token = pro\_item.token\_list[item.dot\_location]

if next\_token.type == 'N':

# 如果下一个token是非终结符，那么可以进行内部的状态位拓展

pro\_id\_list = self.production\_list.get\_by\_char(next\_token.char)

# 获取某个终结符的所有产生式的id

for production\_id in pro\_id\_list:

# 获得用于求first的表达式βa

predict\_list = set()

# 计算新的产生式的预测符号

if pro\_item.length == item.dot\_location + 1:

# β == EPSILON， 直接将alpha作为预测符

predict\_list = item.predict\_list

else:

new\_pro\_item = ProItem()

new\_pro\_item.add(pro\_item.token\_list[item.dot\_location+1:])

# 将预测福序列转化为[char, char\_type]的表达形式

# 求first

predict\_list = self.First(new\_pro\_item)

if EPSILON in predict\_list:

# 当predict\_list有epsilon的时候，再把原来的预测符加上来

predict\_list.extend(item.predict\_list)

lr\_produc\_item = LR\_produc\_item(production\_id, 0, predict\_list)

Status.add(lr\_produc\_item)

new\_length = Status.length

# Status.print\_status()

return Status

* + 下面查看对产生式构造first的算法：
    - 遍历token作为产生式左边时所能产生的所有产生式
    - 获取这些产生式的first()，如果一个token是非终结符，则将递归调用他的first,将结果加入集合set中
    - 查看这个token对应的first的集合是否存在eplison，如果存在，继续往下一个token从第一步开始重复循环
    - 否则结束，返回first构造

def First(self, produc\_item):

"""求first算法

Args:

produc\_item: ProItem,输入的是一个产生式的token\_list

Returns:

predict\_list: 预测符序列

"""

predict\_list = set()

first\_token = produc\_item.get\_token\_by\_index(0)

if first\_token.type == 'T':

predict\_list.add(first\_token.char)

else:

for token in produc\_item.token\_list:

# 遍历产生式的每一个token

first = set()

for produc\_id in self.production\_list.get\_by\_char(token.char):

# 查看当前的token能够产生的所有产生式，produc\_id 为产生式的下标,对产生式求first

first.update(self.First(self.production\_list.get\_right\_by\_index(produc\_id)))

# 遍历完成当前token的所有产生式

# 查看token 的first能否产生SPSILON，如果可以产生，需要把EPSILON去掉，接着求下一个token的first

# 如果不能产生SPSILON，说明遍历已经结束，将得到的SPSILON加入predict\_list

if EPSILON not in first:

predict\_list.update(first)

break

else:

first.remove(EPSILON)

predict\_list.update(first)

return predict\_list

* 下面我们进入processor.goto(status)， 了解状态间拓展的逻辑：
  + 遍历状态中的所有LR产生式
  + 获取产生式的dot的位置，只有还没结尾的产生式，才能进行拓展
  + 将新产生的产生式通过status\_dict字典存储，edge是现在的产生式和新的产生式的链接边：status\_dict[edge] = Status()，这样的结构是为了方便程序返回后进行状态间拓展

"""用来进行状态间拓展

Args:

status 是状态集合

X 是用于goto的边

"""

status\_dict = {}

for lr\_item in status.get\_status\_list():

# item 是一个lr\_produc产生式

pro\_id = lr\_item.produc\_id

# pro\_item 是一个产生式ProItem对象

pro\_item = self.production\_list.get\_right\_by\_index(pro\_id)

if lr\_item.dot\_location < pro\_item.length:

# 这个产生式还没到达末尾，可以进行拓展

# 获得移进的token符号

edge = pro\_item.get\_token\_by\_index(lr\_item.dot\_location).char

# 构造新的lr项

new\_lr\_item = LR\_produc\_item(pro\_id, lr\_item.dot\_location+1, lr\_item.predict\_list)

if status\_dict.has\_key(edge):

# 这个状态已经存在，将新的产生式加入进来

status\_dict[edge].add(new\_lr\_item)

else:

# 这个状态还未出现

status\_dict[edge] = Status()

status\_dict[edge].add(new\_lr\_item)

return status\_dict

## 规约和移进

* 最后，我们回到main函数，对生成的新的状态做以下几个操作：
  + - 填写规约项：processor.regression(definited\_status, new\_status\_id)
    - 填写移进项：processor.add\_to\_parsing\_table('s'+str(new\_status\_id), definited\_status\_id, key)
    - 将新状态插入待拓展队列：to\_extent\_queue.put([new\_status\_id , definited\_status])
* 进一步阐述一下regression(definited\_status, new\_status\_id)规约程序的构造：
  + - 规约程序遍历当前的状态的所有产生式
    - 查看产生式的dot位置
    - 如果产生式的dot已经达到末尾，则对预测分析表填入规约项: self.add\_to\_parsing\_table('r' + str(pro\_id), status\_id, predict\_char)

def regression(self, Status, status\_id):

"""将状态中需要规约的产生式填入predict parsing table

Args：

Status：待检查的状态

status\_id: 这个产生式的id[todo: id应该作为status本身的一个对象，而不是作为参数传递进来]

"""

stat\_list = Status.get\_status\_list()

old\_length = 0

new\_length = Status.length

# 循环直到找不到新的产生式为止

while(old\_length != new\_length):

old\_length = new\_length

for item in stat\_list:

# item 是一个lr\_produc产生式

pro\_id = item.produc\_id

# pro\_item 是一个产生式ProItem对象

pro\_item = self.production\_list.get\_right\_by\_index(pro\_id)

if item.dot\_location == pro\_item.length:

# LR1 产生式的dot已经到末尾了，需要进行规约操作

# print "starting regression"

for predict\_char in item.predict\_list:

self.add\_to\_parsing\_table('r' + str(pro\_id), status\_id, predict\_char)

## 根据预测分析表进行语法分析：

预测分析表的语法分析程序在/main/Parsing.py中，使用ParsingProcessor的类来完成，下面是其主要代码：

* 需要说明的数据结构：
  + 构造输入流，在结尾加上$：self.input\_queue = list(fp.read() + '$')
  + 需要有一个栈，这个栈有两个元素，一个是状态，一个是符号，初始化栈，状态为0号，符号为$

self.token\_stack= ['$']

self.status\_stack = [0]

* + 用于记录分析过程的list：self.parsing\_log = []
  + 传递预测分析表：self.parsing\_table = parsing\_table
  + 传递产生式表：self.production\_list = production\_list
* 主要逻辑过程：
  + 读取token序列
  + 根据预测分析表查看是否有对应的移进/规约项，如果不存在，则语法分析失败
  + 如果结果是移进项，则将状态压入栈，从预测流读取新的元素
  + 如果是规约项
    - 如果规约项是R0，也就是accept，则翻译结束
    - 如果规约项不是r0，则进行弹栈，移进操作

def parsing(self):

"""语法分析的主程序

"""

token = self.input\_queue[0]

# 4. 获取读头的元素token，获取当前的状态号码为status

while(1):

status = self.status\_stack[-1]

# - 查看parsing\_table[status][token]对应的符号，设为 result

# print self.parsing\_table

try:

result = self.parsing\_table[status][token]

except Exception,e:

# - 如果result 为空，分析失败

print "error status: %s ,token: %s"%(status,token)

print "parsing table for this status is : %s"%self.parsing\_table[status]

return False

if result[0]=='s':

# - 如果result 为 移进项，也就是s开头，则获得其移进的项目id，shift\_status\_id

shift\_status\_id = int(result[1:])

# - 将[shift\_status\_id,token]压入栈中，读头读取新的元素

self.add\_log("shift to:%s"%shift\_status\_id)

item =[shift\_status\_id,token]

self.status\_stack.append(shift\_status\_id)

self.token\_stack.append(token)

self.input\_queue.pop(0)

token = self.input\_queue[0]

elif result[0]=='r':

# - 如果result为规约项，也就是r开头

if result[1:] == '0':

# - 如果规约项为r0,那么翻译成功

self.add\_log("accept")

return True

else:

regression\_id = int(result[1:])

pro\_str = self.production\_list.stringtify\_by\_id(regression\_id)

self.add\_log("regression production:%s"%pro\_str)

proitem = self.production\_list.get\_right\_by\_index(regression\_id)

# - 根据规约项产生式的长度，弹出对应的栈，的元素，

length = proitem.length

self.token\_stack = self.token\_stack[:-length]

self.status\_stack = self.status\_stack[:-length]

# - 设规约项左边的符号为token,

new\_token = self.production\_list.get\_left\_by\_index(regression\_id)

# - 根据现在的剩下的栈顶status和新的符号token，求parsing\_table[status][token]对应的符号，设为result

status = self.status\_stack[-1]

# try:

result = self.parsing\_table[status][new\_token]

# except Exception,e:

# return False

# - 因为这里的token为非终结符，所以对应的result一定为移进符号，

# - 将 [result[1:],token]压入栈中

self.status\_stack.append(int(result[1:]))

self.token\_stack.append(new\_token)

# - 产生式规约成功，输出

# Use cases on running

## 输入文件：

在\main\main.py中

production\_fp = open("production.txt",'r') 可以指定输入的语法产生式序列

token\_fp = open("input.txt",'r')可以指定输入的token序列

以下是我的输入文件：

**正确实例：**

S->E

N->N

E->E+T

N->NTN

E->T

N->N

T->T\*F

N->NTN

T->F

N->N

F->(E)

N->TNT

F->i

N->T

注意，文件的末尾需要多空一行

## 输出文件：

通过\main\main.py中的

logging.basicConfig(level=logging.INFO, filename='output.log', filemode='w')

可以对输出进行重定向，我将输出重定向到了’output.log’中，以下是输出结果

**INFO:root:printing the predict parsing table...**

**+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+**

**| STATUS | A:$ | A:) | A:( | A:+ | A:\* | A:i | G:E | G:T | G:F |**

**+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+**

**| 0 | empty | empty | s2 | empty | empty | s1 | s3 | s4 | s5 |**

**| 1 | r6 | empty | empty | r6 | r6 | empty | empty | empty | empty |**

**| 2 | empty | empty | s7 | empty | empty | s6 | s8 | s9 | s10 |**

**| 3 | r0 | empty | empty | s11 | empty | empty | empty | empty | empty |**

**| 4 | r2 | empty | empty | r2 | s12 | empty | empty | empty | empty |**

**| 5 | r4 | empty | empty | r4 | r4 | empty | empty | empty | empty |**

**| 6 | empty | r6 | empty | r6 | r6 | empty | empty | empty | empty |**

**| 7 | empty | empty | s7 | empty | empty | s6 | s13 | s9 | s10 |**

**| 8 | empty | s14 | empty | s15 | empty | empty | empty | empty | empty |**

**| 9 | empty | r2 | empty | r2 | s16 | empty | empty | empty | empty |**

**| 10 | empty | r4 | empty | r4 | r4 | empty | empty | empty | empty |**

**| 11 | empty | empty | s2 | empty | empty | s1 | empty | s17 | s5 |**

**| 12 | empty | empty | s2 | empty | empty | s1 | empty | empty | s18 |**

**| 13 | empty | s19 | empty | s15 | empty | empty | empty | empty | empty |**

**| 14 | r5 | empty | empty | r5 | r5 | empty | empty | empty | empty |**

**| 15 | empty | empty | s7 | empty | empty | s6 | empty | s20 | s10 |**

**| 16 | empty | empty | s7 | empty | empty | s6 | empty | empty | s21 |**

**| 17 | r1 | empty | empty | r1 | s12 | empty | empty | empty | empty |**

**| 18 | r3 | empty | empty | r3 | r3 | empty | empty | empty | empty |**

**| 19 | empty | r5 | empty | r5 | r5 | empty | empty | empty | empty |**

**| 20 | empty | r1 | empty | r1 | s16 | empty | empty | empty | empty |**

**| 21 | empty | r3 | empty | r3 | r3 | empty | empty | empty | empty |**

**+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+**

**INFO:root:printing the definited status list...**

**INFO:root:--------------------------------**

**INFO:root:|status I0: |**

**INFO:root:| S->.E,set(['$']) |**

**INFO:root:| E->.E+T,set(['+', '$']) |**

**INFO:root:| E->.T,set(['+', '$']) |**

**INFO:root:| T->.T\*F,set(['+', '\*', '$']) |**

**INFO:root:| T->.F,set(['+', '\*', '$']) |**

**INFO:root:| F->.(E),set(['$', '+', '\*']) |**

**INFO:root:| F->.i,set(['$', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I1: |**

**INFO:root:| F->i,set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I2: |**

**INFO:root:| F->(.E),set(['+', '\*', '$']) |**

**INFO:root:| E->.E+T,set([')', '+']) |**

**INFO:root:| E->.T,set([')', '+']) |**

**INFO:root:| T->.T\*F,set([')', '+', '\*']) |**

**INFO:root:| T->.F,set([')', '+', '\*']) |**

**INFO:root:| F->.(E),set([')', '+', '\*']) |**

**INFO:root:| F->.i,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I3: |**

**INFO:root:| S->E,set(['$']) |**

**INFO:root:| E->E.+T,set(['+', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I4: |**

**INFO:root:| E->T,set(['+', '$']) |**

**INFO:root:| T->T.\*F,set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I5: |**

**INFO:root:| T->F,set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I6: |**

**INFO:root:| F->i,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I7: |**

**INFO:root:| F->(.E),set([')', '+', '\*']) |**

**INFO:root:| E->.E+T,set([')', '+']) |**

**INFO:root:| E->.T,set([')', '+']) |**

**INFO:root:| T->.T\*F,set([')', '+', '\*']) |**

**INFO:root:| T->.F,set([')', '+', '\*']) |**

**INFO:root:| F->.(E),set([')', '+', '\*']) |**

**INFO:root:| F->.i,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I8: |**

**INFO:root:| F->(E.),set(['+', '\*', '$']) |**

**INFO:root:| E->E.+T,set([')', '+']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I9: |**

**INFO:root:| E->T,set([')', '+']) |**

**INFO:root:| T->T.\*F,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I10: |**

**INFO:root:| T->F,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I11: |**

**INFO:root:| E->E+.T,set(['+', '$']) |**

**INFO:root:| T->.T\*F,set(['+', '\*', '$']) |**

**INFO:root:| T->.F,set(['+', '\*', '$']) |**

**INFO:root:| F->.(E),set(['$', '+', '\*']) |**

**INFO:root:| F->.i,set(['$', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I12: |**

**INFO:root:| T->T\*.F,set(['+', '\*', '$']) |**

**INFO:root:| F->.(E),set(['$', '+', '\*']) |**

**INFO:root:| F->.i,set(['$', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I13: |**

**INFO:root:| F->(E.),set([')', '+', '\*']) |**

**INFO:root:| E->E.+T,set([')', '+']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I14: |**

**INFO:root:| F->(E),set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I15: |**

**INFO:root:| E->E+.T,set([')', '+']) |**

**INFO:root:| T->.T\*F,set([')', '+', '\*']) |**

**INFO:root:| T->.F,set([')', '+', '\*']) |**

**INFO:root:| F->.(E),set([')', '+', '\*']) |**

**INFO:root:| F->.i,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I16: |**

**INFO:root:| T->T\*.F,set([')', '+', '\*']) |**

**INFO:root:| F->.(E),set([')', '+', '\*']) |**

**INFO:root:| F->.i,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I17: |**

**INFO:root:| E->E+T,set(['+', '$']) |**

**INFO:root:| T->T.\*F,set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I18: |**

**INFO:root:| T->T\*F,set(['+', '\*', '$']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I19: |**

**INFO:root:| F->(E),set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I20: |**

**INFO:root:| E->E+T,set([')', '+']) |**

**INFO:root:| T->T.\*F,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:--------------------------------**

**INFO:root:|status I21: |**

**INFO:root:| T->T\*F,set([')', '+', '\*']) |**

**INFO:root:--------------------------------**

**INFO:root:**

**INFO:root:printing the parsing step...**

**+----+----------------+----------------------+--------------------------------+------------------------------+**

**| id | status stack | token stack | input | action |**

**+----+----------------+----------------------+--------------------------------+------------------------------+**

**| 0 | [0] | ['$'] | ['i', '\*', 'i', '+', 'i', '$'] | shift to:1 |**

**| 1 | [0, 1] | ['$', 'i'] | ['\*', 'i', '+', 'i', '$'] | regression production:F->i |**

**| 2 | [0, 5] | ['$', 'F'] | ['\*', 'i', '+', 'i', '$'] | regression production:T->F |**

**| 3 | [0, 4] | ['$', 'T'] | ['\*', 'i', '+', 'i', '$'] | shift to:12 |**

**| 4 | [0, 4, 12] | ['$', 'T', '\*'] | ['i', '+', 'i', '$'] | shift to:1 |**

**| 5 | [0, 4, 12, 1] | ['$', 'T', '\*', 'i'] | ['+', 'i', '$'] | regression production:F->i |**

**| 6 | [0, 4, 12, 18] | ['$', 'T', '\*', 'F'] | ['+', 'i', '$'] | regression production:T->T\*F |**

**| 7 | [0, 4] | ['$', 'T'] | ['+', 'i', '$'] | regression production:E->T |**

**| 8 | [0, 3] | ['$', 'E'] | ['+', 'i', '$'] | shift to:11 |**

**| 9 | [0, 3, 11] | ['$', 'E', '+'] | ['i', '$'] | shift to:1 |**

**| 10 | [0, 3, 11, 1] | ['$', 'E', '+', 'i'] | ['$'] | regression production:F->i |**

**| 11 | [0, 3, 11, 5] | ['$', 'E', '+', 'F'] | ['$'] | regression production:T->F |**

**| 12 | [0, 3, 11, 17] | ['$', 'E', '+', 'T'] | ['$'] | regression production:E->E+T |**

**| 13 | [0, 3] | ['$', 'E'] | ['$'] | accept |**

**+----+----------------+----------------------+--------------------------------+------------------------------+**

# Problems occurred and related solutions

* 1. 本试验的难点在两个方面：
     1. 数据结构的设计
     2. 几个关键算法的实现

通过仔细阅读课程教材，最后比较完善地解决了这个问题

# Your feelings and comments

LR(1)的语法分析器基本已经实现，并且有了比较可视化的结果，接下来的工作将是：

* + 增加二义性文法的分析功能
  1. 与词法分析器结合，完成从输入字符串到生成语法分析序列的整个过程。

项目源代码已经上传到github上，我将继续完善这个项目：

<https://github.com/Bunrning-Bear/compiler_experiment_python>