编译原理实验六: 对表达式的解析

2023-12-23 13:43:50

## 实验目的：

熟悉ply的基本使用，能够通过编写无二义性的语法以及yacc来实现对化学分子式的基本解析

## 实验内容：

编写程序并且对化学分子式进行解析，得到其内部的元素及其元素个数

以下是例子：

atom\_count("He") == 1  
atom\_count("H2") == 2  
atom\_count("H2SO4") == 7  
atom\_count("CH3COOH") == 8  
atom\_count("NaCl") == 2  
atom\_count("C60H60") == 120

## 思路：

首先需要进行词法分析的构造

### 词法分析

在本次实验中我们显然只需要对化学元素以及数字进行解析。事实上在一个化学分子式中确实不会出现除这两者之外的其它事物

* 对化学元素的解析：在给定的化学元素周期表中，只要给出一个恰好包括所有元素的正则表达式即可
* 对数字的解析：不难得到

由此我们给出词法分析的代码：

# List of token names. This is always required  
tokens = (  
 # 本次实验我们只需要解析以下两个类型的词语  
 'NUMBER',# 数字  
 'SYMBOL' # 化学元素  
)  
  
  
# A regular expression rule with some action code  
def t\_NUMBER(t):#对数字的识别  
 # 显然  
 r'\d+'  
 t.value = int(t.value)  
 return t  
  
def t\_SYMBOL(t):# 对元素的识别  
 # 由元素周期表可得  
 r"""  
 C[laroudsemf]?|Os?|N[eaibdpos]?|S[icernbmg]?|P[drmtboau]?|  
 H[eofgas]?|A[lrsgutcm]|B[eraik]?|Dy|E[urs]|F[erm]?|G[aed]|  
 I[nr]?|Kr?|L[iaur]|M[gnodt]|R[buhenaf]|T[icebmalh]|  
 U|V|W|Xe|Yb?|Z[nr]  
 """  
  
 return t;  
  
# A string containing ignored characters (spaces and tabs)  
t\_ignore = ' \t'  
  
# Error handling rule  
def t\_error(t):  
 print ("Illegal character '%s'" % t.value[0])  
 t.lexer.skip(1)  
  
## Build the lexer  
lexer = lex.lex()

### 语法

考虑的基本语法如下：

species\_list -> species\_list species  
species\_list -> species  
species -> SYMBOL  
species -> SYMBOL COUNT

其中表示一个化学分子式，表示一个基本的化学元素的表达式，表示能够识别出来的化学元素，就表示对应的次数，一个表达式由上述两者组成

此外，为了让最后的结果能够对不同元素的次数加以区分，我们建立了一个结构体来方便计数

class Atom(object):  
 def \_\_init\_\_(self, symbol, count):  
 self.symbol = symbol  
 self.count = count  
 def \_\_repr\_\_(self):  
 return "Atom(%r, %r)" % (self.symbol, self.count)

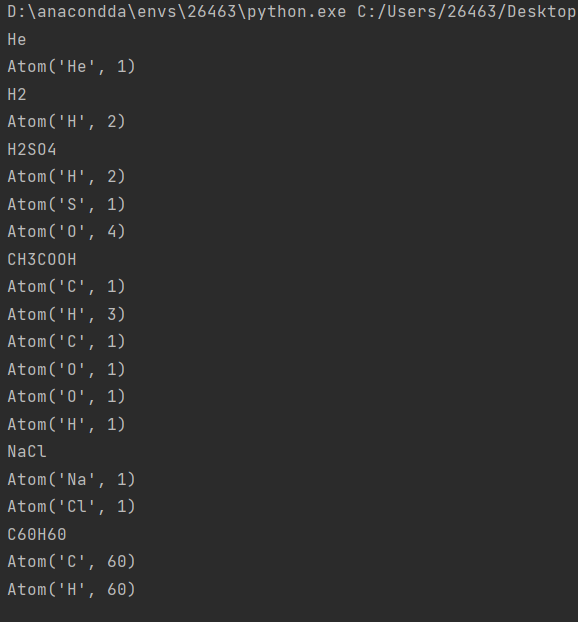
从而不难得到语法如下

# 给出的语法如下，用''包装  
# 第一条默认为初始语法  
def p\_splist\_sp(p):  
 'splist : splist sp'  
 p[1].append(p[2])  
 p[0]=p[1]  
  
def p\_splist\_to\_sp(p):  
 'splist : sp'  
 p[0] = [p[1]]  
  
def p\_sp\_symbol(p):  
 'sp : SYMBOL'  
 p[0] = Atom(p[1],1)  
  
def p\_sp\_symbol\_count(p):  
 'sp : SYMBOL NUMBER'  
 p[0] = Atom(p[1],p[2])  
  
# Error rule for syntax errors  
def p\_error(p):  
 print("Syntax error in input!")

当单独一个或者与规约为一个时，我们直接将其包装为一个基本结构体。否则用一个以Atom为基本元素的数组来记录其内包含的元素及其个数

yacc采用LALR分析法，当我们编译这个程序的时候，就会在同一个目录下得到LALR生成的分析表parser.out，并且之后会根据其中的DFA来决定分析时的移进/归约操作

## 实验结果&改进



不过注意到这个记录方法还是存在一定不足，在CH3COOH中C出现了两次，但是两次分开记录了，最好可以合并在一起

由此我们可以采用字典来记录

以下是chemistry\_caculate.py的改进后的语法

#! /usr/bin/env python  
# coding=utf-8  
import ply.yacc as yacc  
from calclex import tokens  
  
# rules for species  
def p\_splist\_sp(p):  
 'splist : splist sp'  
 for sym,cnt in p[2].items():  
 if sym not in p[1]:  
 p[1][sym]=cnt  
 else:  
 p[1][sym]+=cnt  
 p[0]=p[1]  
  
def p\_splist\_to\_sp(p):  
 'splist : sp'  
 p[0] = p[1]  
  
def p\_sp\_symbol(p):  
 'sp : SYMBOL'  
 p[0] = {}  
 p[0][p[1]] = 1  
  
def p\_sp\_symbol\_count(p):  
 'sp : SYMBOL NUMBER'  
 p[0] = {}  
 p[0][p[1]] = p[2]  
  
# Error rule for syntax errors  
def p\_error(p):  
 print("Syntax error in input!")

此时结果如下：

可以看到确实已经合并在一起了