# 词法分析器-Python版

## 大纲

## 目标

本次实验的主要目的是对自定义的程序语言的词法分析器程序构造，我从Python语言当中选择了部分具有代表性的子集，实现词法分析器，主要是对编译原理课 程中学习的从正则表达式转化为 NFA，再从 NFA 转化为 DFA 以及后续的代码生成的过程有 更深刻的认识。同时，也希望对于在编译原理课程中所体现出的计算机科学当中的一些朴素 而优美的思想有更多的体会。

## 内容概述

首先确定了Python的关键字以及符号并编码，之后幸运的找到了Python自己实现的词法分析工具tokenize中的generate_token方法。并以他为目标，实现了自己的token分析

## 假设与依赖

无恶意代码，因为目标实现了多行注释等识别，但是对于转义字符还没有（无刻意刁难就好）

## 记号定义

| word | id | synax | id |
|---|---|---|---|
| False | 1 | = | 34 |
| class | 2 | / | 35 |
| finally | 3 | + | 36 |
| is | 4 | - | 37 |
| return | 5 | * | 38 |
| None | 6 | ! | 39 |
| continue | 7 | # | 40 |
| for | 8 | % | 41 |
| lambda | 9 | < | 42 |
| try | 10 | > | 43 |
| True | 11 | ^ | 44 |
| def | 12 | ~ | 45 |
| from | 13 | ( | 46 |
| nonlocal | 14 | ) | 47 |
| while | 15 | [ | 48 |
| and | 16 | ] | 49 |
| del | 17 | { | 50 |

| | | | |
|---|---|---|---|
| global | 18 | } | 51 |
| not | 19 | ' | 52 |
| with | 20 | " | 53 |
| as | 21 | : | 54 |
| elif | 22 | ; | 55 |
| if | 23 | ''' | 56 |
| or | 24 | == | 57 |
| yield | 25 | != | 58 |
| assert | 26 | += | 59 |
| else | 27 | -= | 60 |
| import | 28 | /= | 61 |
| pass | 29 | *= | 62 |
| break | 30 | %= | 63 |
| except | 31 | >> | 64 |
| in | 32 | << | 65 |
| raise | 33 | >= | 66 |
| | | <= | 67 |

## 思路与方法

python的好处在于一行基本只有一段代码，因此划分比较简单。之后就按照自动机理论去匹配即可。关于各种自动机的示例在下面一节。匹配分析后，会生成表格，记录token的代号，以及位置等信息。

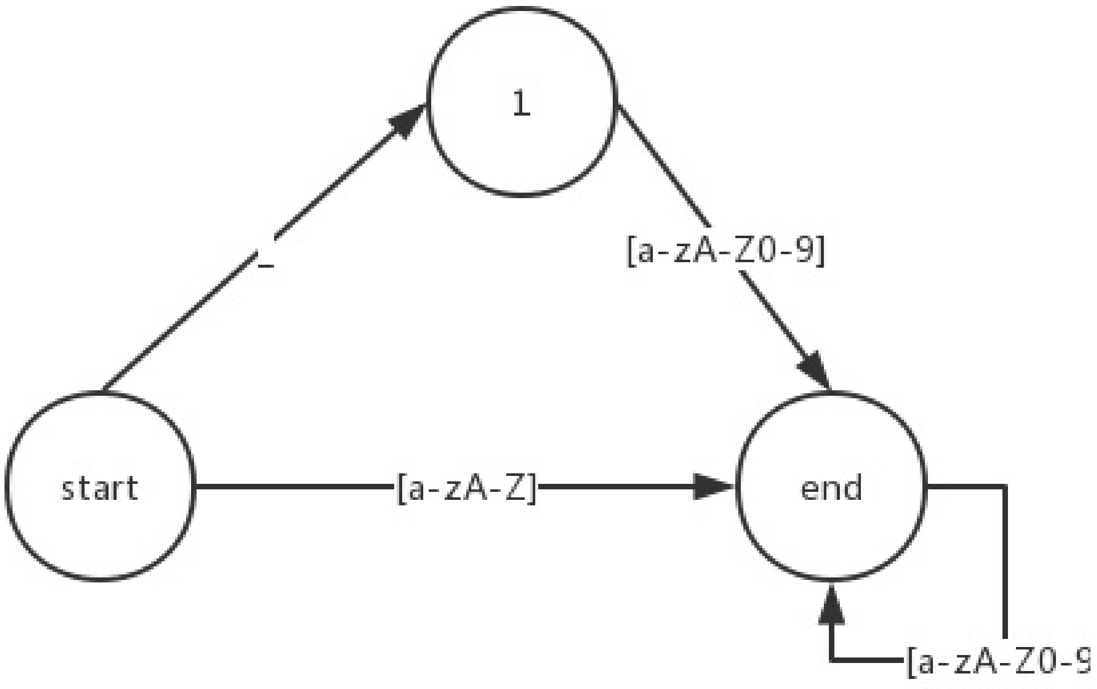
其中解析多行注释或字符花了一些功夫，因为不知道有几行。。。。 最后是采用上层模块识别的方法。即机器本身也对'''之间的词进行解析，只不过

忽视结果直接拼接，直到下一个'''出现

## 有限自动机

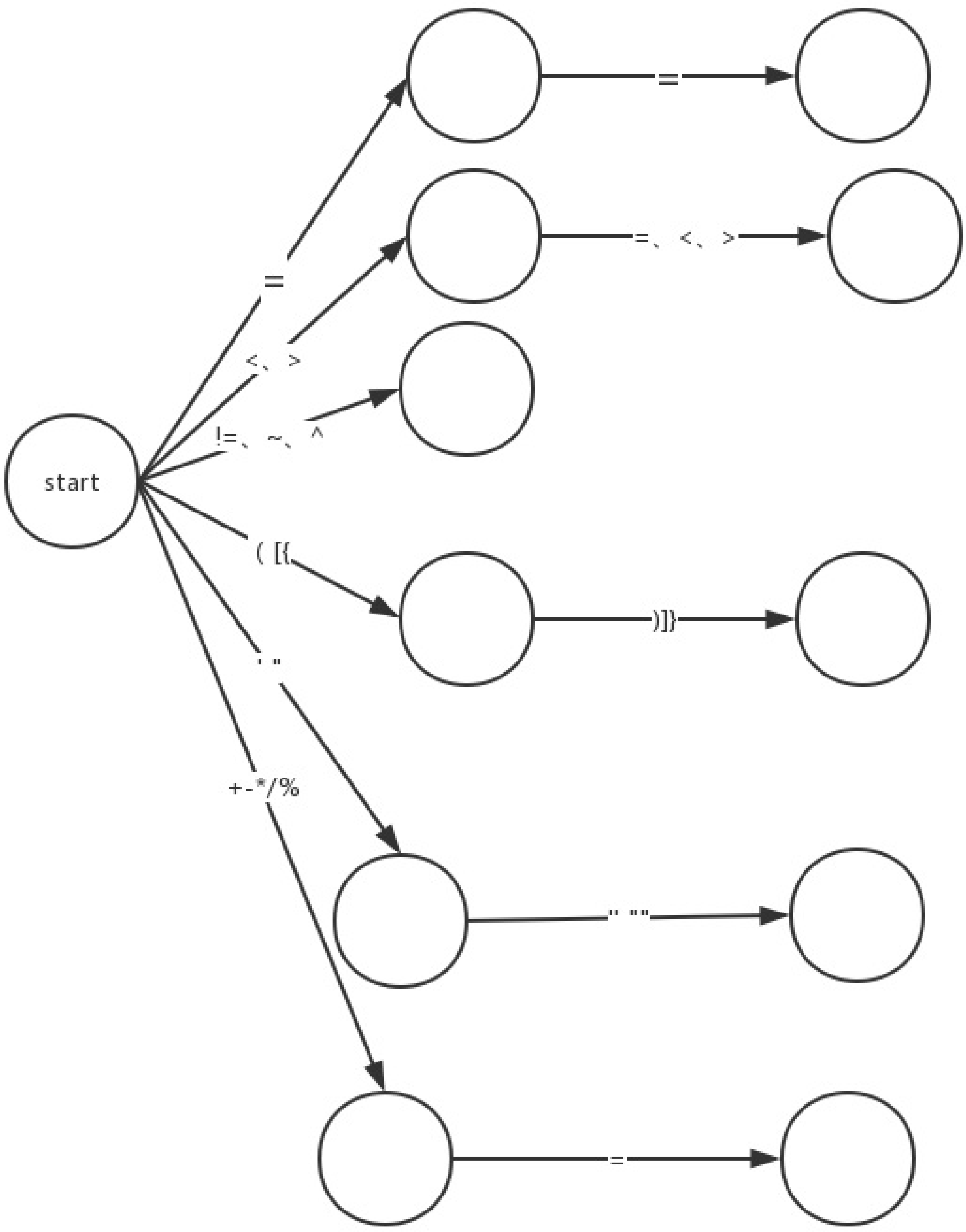
我们根据python的语法特性构造了字符、符号以及数字的有限自动机，分别如下图所示:

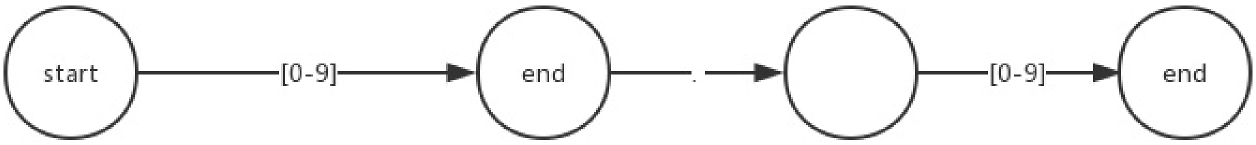
字符自动机



符号自动机(其中空白圆圈均为终止状态)

数字自动机





## 测试用例与检验

测试用例为：

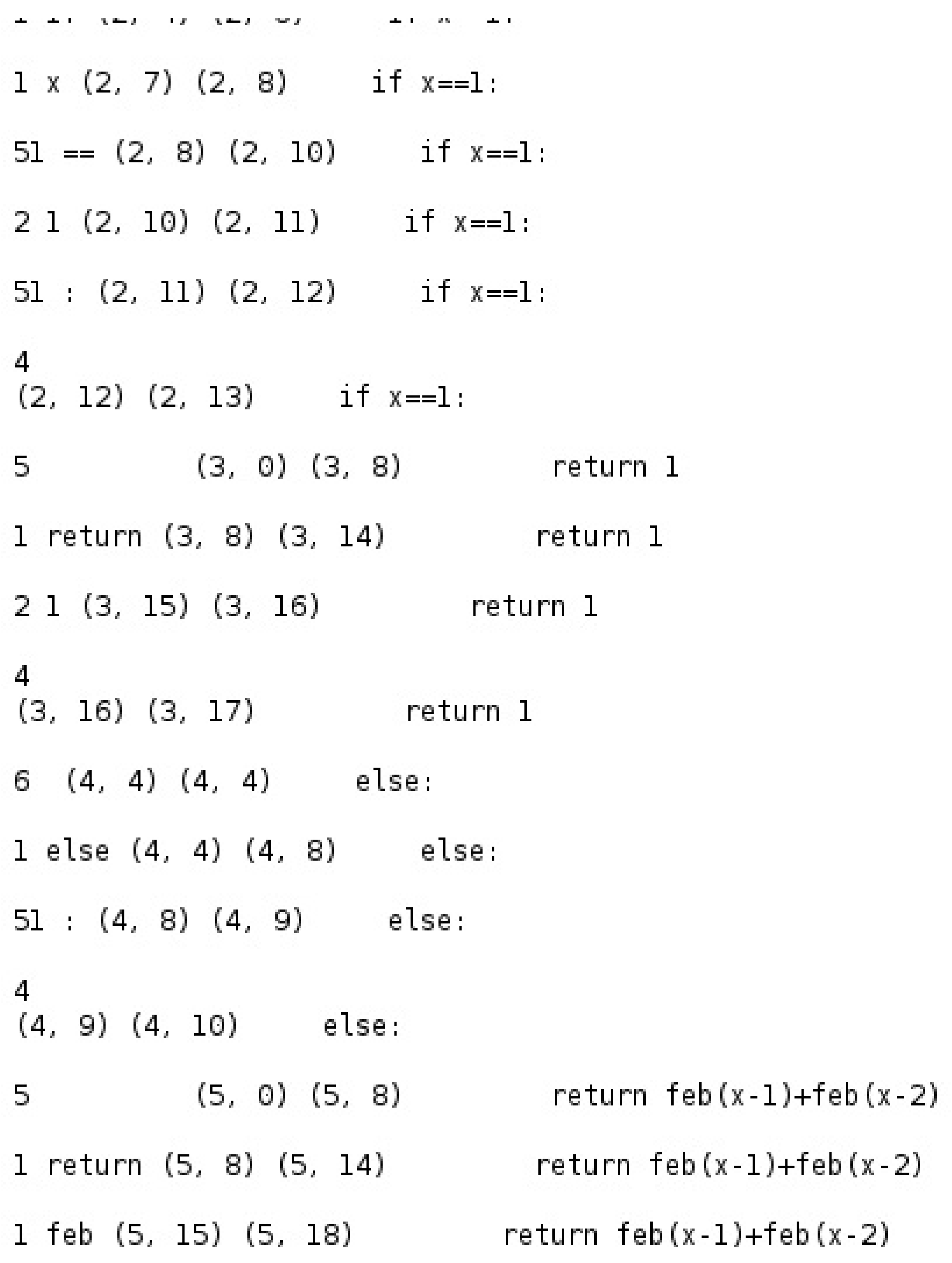
```
def feb(x):
    if x==1:
        return 1
    else:
        return feb(x-1)+feb(x-2)

def do():
    for i in range(1,200):
        print 'num is '+i
str='''
haha
lalala
'''
feb(lambda x:x+1)
```

我的代码的输出为：

```
          val  id  line  startPosition  endPosition
0         def  12     1              0            4
1         feb  69     1              4            8
2           (  46     1              7            8
3           x  70     1              8           10
4           )  47     1              9           10
5           :  55     1             10           11
6          if  23     2              4            7
7           x  70     2              7            9
8          ==  58     2              8           10
9           1  71     2             10           11
10          :  55     2             11           12
11     return   5     3              8           15
12          1  71     3             15           16
13       else  27     4              4            9
14          :  55     4              8            9
15     return   5     5              8           15
16        feb  69     5             15           19
17          (  46     5             18           19
18          x  70     5             19           21
19          -  37     5             20           21
20          1  71     5             21           22
21          )  47     5             22           23
22          +  36     5             23           24
23        feb  69     5             24           28
24          (  46     5             27           28
25          x  70     5             28           30
26          -  37     5             29           30
27          2  72     5             30           31
28          )  47     5             31           32
29        def  12     7              0            4
```

generate_token输出为：

```
1 def (1, 0) (1, 3) def feb(x):

1 feb (1, 4) (1, 7) def feb(x):

51 ( (1, 7) (1, 8) def feb(x):

1 x (1, 8) (1, 9) def feb(x):

51 ) (1, 9) (1, 10) def feb(x):

51 : (1, 10) (1, 11) def feb(x):

4
(1, 11) (1, 12) def feb(x):

5       (2, 0) (2, 4)     if x==1:

1 if (2, 4) (2, 6)     if x==1:
```



```
1 x (2, 7) (2, 8)     if x==1:

51 == (2, 8) (2, 10)     if x==1:

2 1 (2, 10) (2, 11)     if x==1:

51 : (2, 11) (2, 12)     if x==1:

4
(2, 12) (2, 13)     if x==1:

5          (3, 0) (3, 8)         return 1

1 return (3, 8) (3, 14)         return 1

2 1 (3, 15) (3, 16)         return 1

4
(3, 16) (3, 17)         return 1

6  (4, 4) (4, 4)     else:

1 else (4, 4) (4, 8)     else:

51 : (4, 8) (4, 9)     else:

4
(4, 9) (4, 10)     else:

5          (5, 0) (5, 8)         return feb(x-1)+feb(x-2)

1 return (5, 8) (5, 14)         return feb(x-1)+feb(x-2)

1 feb (5, 15) (5, 18)         return feb(x-1)+feb(x-2)
```

## 心得与体会

本次实践了词法分析的最简单的形式，感觉编译之路真的是博大精深。最大的收获其实是通过阅读generate_token源码获得的，虽然我的实现要丑陋的多。因此，下面贴出代码，以示尊敬

```
def generate_tokens(readline):
    """
    The generate_tokens() generator requires one argument, readl
    must be a callable object which provides the same interface 
    readline() method of built-in file objects. Each call to the
    should return one line of input as a string.  Alternately, r
    can be a callable function terminating with StopIteration:
        readline = open(myfile).next    # Example of alternate r

    The generator produces 5-tuples with these members: the toke
    token string; a 2-tuple (srow, scol) of ints specifying the 
    column where the token begins in the source; a 2-tuple (erow
    ints specifying the row and column where the token ends in t
    and the line on which the token was found. The line passed i
    logical line; continuation lines are included.
    """
    lnum = parenlev = continued = 0
    namechars, numchars = string.ascii_letters + '_', '012345678
    contstr, needcont = '', 0
    contline = None
    indents = [0]

    while 1:                                  # loop over lines
        try:
            line = readline()
        except StopIteration:
            line = ''
        lnum += 1
        pos, max = 0, len(line)

        if contstr:                           # continued strin
            if not line:
                raise TokenError, ("EOF in multi-line string", s
            endmatch = endprog.match(line)
            if endmatch:
                pos = end = endmatch.end(0)
                yield (STRING, contstr + line[:end],
                       strstart, (lnum, end), contline + line)
                contstr, needcont = '', 0
                contline = None
            elif needcont and line[-2:] != '\\\n' and line[-3:]
```

```
                yield (ERRORTOKEN, contstr + line,
                           strstart, (lnum, len(line)), contline
                contstr = ''
                contline = None
                continue
            else:
                contstr = contstr + line
                contline = contline + line
                continue

        elif parenlev == 0 and not continued:  # new statement
            if not line: break
            column = 0
            while pos < max:                   # measure leading
                if line[pos] == ' ':
                    column += 1
                elif line[pos] == '\t':
                    column = (column//tabsize + 1)*tabsize
                elif line[pos] == '\f':
                    column = 0
                else:
                    break
                pos += 1
            if pos == max:
                break

            if line[pos] in '#\r\n':           # skip comments o
                if line[pos] == '#':
                    comment_token = line[pos:].rstrip('\r\n')
                    nl_pos = pos + len(comment_token)
                    yield (COMMENT, comment_token,
                           (lnum, pos), (lnum, pos + len(comment
                    yield (NL, line[nl_pos:],
                           (lnum, nl_pos), (lnum, len(line)), li
                else:
                    yield ((NL, COMMENT)[line[pos] == '#'], line
                           (lnum, pos), (lnum, len(line)), line
                continue
            if column > indents[-1]:           # count indents o
                indents.append(column)
                yield (INDENT, line[:pos], (lnum, 0), (lnum, pos
            while column < indents[-1]:
                if column not in indents:
                    raise IndentationError(
                        "unindent does not match any outer inden
                        ("<tokenize>", lnum, pos, line))
                indents = indents[:-1]
                yield (DEDENT, '', (lnum, pos), (lnum, pos), lin

        else:                                  # continued state
            if not line:
                raise TokenError, ("EOF in multi-line statement"
            continued = 0

        while pos < max:
            pseudomatch = pseudoprog.match(line, pos)
            if pseudomatch:                                # sca
                start, end = pseudomatch.span(1)
                spos, epos, pos = (lnum, start), (lnum, end), en
                if start == end:
                    continue
                token, initial = line[start:end], line[start]

                if initial in numchars or \
                   (initial == '.' and token != '.'):      # ord
                    yield (NUMBER, token, spos, epos, line)
                elif initial in '\r\n':
                    yield (NL if parenlev > 0 else NEWLINE,
                           token, spos, epos, line)
                elif initial == '#':
                    assert not token.endswith("\n")
                    yield (COMMENT, token, spos, epos, line)
                elif token in triple_quoted:
                    endprog = endprogs[token]
                    endmatch = endprog.match(line, pos)
                    if endmatch:                           # all
                        pos = endmatch.end(0)
                        token = line[start:pos]
                        yield (STRING, token, spos, (lnum, pos),
                    else:
                        strstart = (lnum, start)           # mul
                        contstr = line[start:]
                        contline = line
                        break
                elif initial in single_quoted or \
                    token[:2] in single_quoted or \
                    token[:3] in single_quoted:
                    if token[-1] == '\n':                  # con
                        strstart = (lnum, start)
                        endprog = (endprogs[initial] or endprogs
                                   endprogs[token[2]])
                        contstr, needcont = line[start:], 1
                        contline = line
                        break
                    else:                                  # ord
                        yield (STRING, token, spos, epos, line)
                elif initial in namechars:                 # ord
                    yield (NAME, token, spos, epos, line)
                elif initial == '\\':                      # con
                    continued = 1
                else:
                    if initial in '([{':
                        parenlev += 1
                    elif initial in ')]}':
                        parenlev -= 1
                    yield (OP, token, spos, epos, line)
            else:
                yield (ERRORTOKEN, line[pos],
                           (lnum, pos), (lnum, pos+1), line)
                pos += 1

    for indent in indents[1:]:                 # pop remaining i
        yield (DEDENT, '', (lnum, 0), (lnum, 0), '')
    yield (ENDMARKER, '', (lnum, 0), (lnum, 0), '')
```