Lab 5

Monday, September 26, 2022 10:44 AM

1. Implement the parser for the following grammar:

<S> → <A><B><C>	{'a', 'b', 'c', EOF}
<A> → 'a'	{'a'}
<A> → λ	{'b', 'c', EOF}
<B> → 'b'	{'b'}
<B> → λ	{'c', EOF}
<C> → 'c'	{'c'}
<C> → λ	{EOF}

Test your parser with abc, ac, bc, a, b, c, λ, aaa, abcd, aba, and acb.

```
def S():
    if token == 'a' or token == 'b' or token == 'c' or token == '':
        A()
        B()
        C()
    else:
        raise RuntimeError ('expecting a, b, or c ')

def A():
    if token == 'a':
        advance()
    elif token == 'b' or token == 'c' or token == '':
        pass
    else:
        raise RuntimeError ('Expecting a or b')

def B():
    if token == 'b':
        advance()
    elif token == 'c' or token == '':
        pass
    else:
        raise RuntimeError ("expecting b or c")

def C():
   if token == 'c':
      # perform actions corresponding to production 3
      advance()
   elif token == '':
      pass
   else:
      raise RuntimeError('Expecting c')
```

```
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py abc
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py ac
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py bc
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py a
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py b
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py c
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py ''
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py aaa
expecting b or c
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py abcd
Garbage following <S>-string
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py aba
Expecting c
beesannekurzum@client181-110 Lab 5 % python3 Lab5.py acb
Garbage following <S>-string
```

3. Determine the predict sets for the following grammar:

```
<S> → <A><B>'d'
<A> → 'a'<A>'c'
<A> → λ
<B> → 'b'<A>'e'
<B> → λ
```

Predict sets

<S> → <A><B>'d'	{'a', 'b', 'd'}
<A> → 'a'<A>'c'	{'a'}
<A> → λ	{'b', 'd', 'c', 'e'}
<B> → 'b'<A>'e'	{'b'}
<B> → λ	{'d'}

<S> → 'c'

9. Redo problem 1, but do not use predict sets. Instead, use a *backtracking* recursive-descent parser. Use spbt.py (the backtracking version of sp.py) as your model. Here is the essential idea behind the backtracking approach: Suppose there are five <A> productions. Then your parser should have an A() function and five additional functions—A1(), A2(), A3(), A4(), and A5()—one for each of the five <A> productions. Each of these five additional functions applies the right side of the corresponding production. If it succeeds, it returns True. Otherwise, it returns False. When the A() function is called, it saves the current position in the token stream. It then calls A1(). If A1() returns True, then A() returns True to its caller. However, if A1() returns False, then A() resets the current position in the token stream to the saved position. It then calls A2(). If A2() returns True, then A() returns True to its caller. If, however, A2() returns False, then A() again resets the current position in the token stream to the saved position. It then calls A3(). A() continues in this fashion, trying each production by calling the corresponding function until one returns True. If they all return False, then A() returns False to its caller. To distinguish this type of recursive-descent parser from the predictive type described in this chapter, we call this type a *backtracking recursive-descent parser*.

```
def S():
   return A() and B() and C()
def A():
    global tokenindex
    save = tokenindex
    if A1():
        return True
    else:
        tokenindex = save
        return A2()
def A1():
    return consume('a')
def A2():
    return True
def B():
    global tokenindex
    save = tokenindex
    if B1():
        return True
    else:
        tokenindex = save
        return B2()
def B1():
    return consume('b')
def B2():
    return True
def C():
    global tokenindex
    save = tokenindex
    if C1():
        return True
    else:
        tokenindex = save
        return C2()
def C1():
    return consume('c')
def C2():
    return True
```

```
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py abc
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py ac
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py bc
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py a
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py b
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py c
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py ''
String in language
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py aaa
Garbage following string
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py abcd
Garbage following string
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py aba
Garbage following string
beesannekurzum@client181-110 Lab 5 % python3 Lab5Q9.py acb
Garbage following string
```