DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Super Collider

16 / 12 / 2013 Teoría de Lenguajes

Grupo 12

Integrante	LU	Correo electrónico
Carreiro, Martin	45/10	martin301290@gmail.com
Kujawski, Kevin	459/10	kevinkuja@gmail.com
Ortiz De Zarate, Juan Manuel	403/10	jmanuoz@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Introduccíón

El enunciado plantea la necesidad de generar un parsear para una gramatica y generar sonido a partir de la misma. Para eso usamos la misma gramatica que para el primer TP ya que consideramos correcta y acorde para resolver esta necesidad, y si bien hay conflictos SHIFT/REDUCE en el siguiente estado:

```
s -> s o s .
s -> s . POINT p
s -> s . o s
o -> . CON
o -> . MIX
o -> . ADD
o -> . SUB
o -> . MUL
o -> . DIV
```

El parser YACC decide correctamente hacer SHIFT ya que hacer un REDUCE carecería de sentido

La solución se implemento en Python usando PLY, el cual contiene a YACC que usa la tecnica LALR para el analisis sintactico.

1.1. Manual de uso del Super Collider

El Super Collider tal como esta pensado para generar buffers de sonidos en tiempo real tiene un lenguaje el cual su uso se pensó para que sea simple y concreto. Su escritura se divide en tres grupos principales: generadores, metodos y operadores. Cuyo resultado siempre es una "tira"de numeros concatenados.

Los generadores se pueden pensar como valores concretos para comenzar, a lo que luego se le aplican los metodos y operadores. Estos son sin, lin, sil, noi y numeros decimales (con o sin signo)

Los operadores actuan sobre dos buffers, operando entre ellos y generando uno nuevo como resultado. Estos son con, mix, add, sub, mul y div.

Los metodos son el grupo más relacionado con la parte musical, ya que son los encargados de, partiendo de un buffer, transformarlo aplicando operaciones puramente de manipulación de sonido y a su vez pueden contener efectos externos tales como imprimir buffer o reproducir sonido. Estos son play, post, loop, tune, fill, reduce, expand.

Expliquemos un poco su uso, suponiendo que B es un buffer, OP una operador, G un generador y M un metodo. Entonces sus posibles usos son:

Generadores, se utilizan solos, algunos pueden requerir parametros: G

Metodos, a un buffer se le aplica un metodo que genera otro buffer nuevo B.M

Operadores, realiza operaciones sobre 2 buffers y generan otro nuevo. B1 OP B2

1.2. Manual de uso del interprete

Primero se debe generar la gramatica con "python main.py". Luego para usar el parser hay dos funciones en el main.py, parsearArchivo(ruta) y parsearCadena(cadena), la cual se le pasa una ruta y una cadena respectivamente. En el caso del metodo parsearArchivo, el archivo del mismo debera contener una cadena sola, la cual puede tener comentarios (usando //) los cuales se ignoraran y saltos de linea los cuales se concatenaran al cargar el archivo.

1.3. Librerias necesarias para PYTHON

PLY, PYGAMES

1.4. Decisiones y aclaraciones

Por como es el analizador sintactico, por un tema de precedencia, decidimos que los operadores (add, sub, mix, con, div, mul) se van evaluando de derecha a izquierda, y si se quiere evitar esto se deben usar entre llaves. Es decir, E OP E debe ir como E OP E si se quiere imponer un orden de evaluación. Esta decision la tomamos debido a que por como se realiza la recursión necesitamos la expresividad de las llaves para poder decidir en que orden se ejecutan los operadores.

Ejemplos:

2+2;2;2 = 4;4;4

2;2;2+2 = 2;4;4

Además, aunque hay funciones cuyas especificaciones requieren parametros, en algunos metodos brindamos la posibilidad de no agregarlos y tener valores por defecto.

El audio se genera a 44.1kHz, 16-bit signed, mono

1.5. Ejemplos permitidos

- { {{-1;1}.loop(44).expand(2).tune(-1).loop(2.9).fill(3)}.loop(5); {-1;1}.loop(44).expand(2).loop(6); {-1;1}.loop(44).expand(
  {-1;1}.loop(44).expand(2).tune(5).loop(3).fill(3); {-1;1}.loop(44).expand(2).tune(-1).loop(3).fill(3); {-1;1}.loop(44).expand
  {-1;1}.loop(44).expand(2).tune(-1).loop(4.5).fill(4.5); {-1;1}.loop(44).expand(2).tune(-5).loop(12).fill(12); {{-1;1}.loop(44)
  {-1;1}.loop(44).expand(2).tune(-3).loop(2.9).fill(3); {-1;1}.loop(44).expand(2).loop(4.5); {-1;1}.loop(44).expand(2).tune(-
  5).loop(5.9).fill(6); {-1;1}.loop(44).expand(2).tune(-5).loop(3).fill(3); {-1;1}.loop(44).expand(2).loop(3); {-1;1}.loop(44).ex
  {-1;1}.loop(44).expand(2).tune(+4).loop(2.9).fill(3); {-1;1}.loop(44).expand(2).loop(8.9).fill(9) }.loop(3).post()
- {-2.loop(4);{2-1};3}.post()
- {sin(34,45).tune(2);sin(3,1).tune(1)}.post().play(500)
- {-1;1}.post().loop(44).expand(12).post()
- {-1;1}.loop(44).expand(2).tune(15).play(500).post
- {1}

1.6. Ejemplos no permitidos

- 1; operadores necesitan dos buffers
- 2.play faltan parámetros
- .loop(15) falta buffers
- {2}.loop({9}) pasa buffer como parametro y no un número

2. Código

2.1. Lexer

```
tokens = (
    'CON', 'MIX', 'ADD', 'SUB', 'MUL', 'DIV', 'SIN',
    'LIN', 'NOI', 'SIL', 'PLAY', 'POST', 'LOOP',
        'TUNE', 'FILL', 'REDUCE', 'EXPAND', 'LKEY', 'RKEY',
        'LPAREN', 'RPAREN', 'POINT', 'COMA',  'FLOAT'
    )

# Tokens

t_CON     = r'con|;'
t_MIX    = r'mix|&'
t_ADD    = r'add|\+'
t_SUB   = r'sub|-'
t_MUL   = r'mul|\*'
t_DIV   = r'div|/'
t_SIN   = r'sin'

t_LIN   = r'lin|linear'
t_NOI   = r'noi|noise'
t_SIL   = r'sil|silence'
t_PLAY   = r'play'
t_POST   = r'post'
t_LOOP   = r'loop'
t_TUNE   = r'tune'
t_FILL   = r'fill'
t_REDUCE   = r'reduce'
t_EXPAND   = r'expand'
t_LPAREN   = r'\('
t_RPAREN   = r'\)'
t_LKEY   = r'\{'
t_RKEY   = r'\}'
t_POINT   = r'\.'
t_COMA = r','
```

```
def t_FLOAT(t):
    r'(\d+\.\d+|\d+)'
    try:
        t.value = float(t.value)
    except ValueError:
        print("Float value too large %d", t.value)
        t.value = 0
    return t

# Ignored characters
t_ignore = " \t"

def t_newline(t):
    r'\n+'
    t.lexer.lineno += t.value.count("\n")

def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

# Build the lexer
import ply.lex as lex
lex.lex()
```

2.2. Parser

```
from tokens import *
from funciones import *

precedence = (
    ('right','UMINUS'),
    ('right','SUB','CON'),
    )

def p_s1(t):
        '''s : g'''
        t[0] = t[1]
def p_s2(t):
        '''s : s POINT p'''
        t[0] = method(t[3]['method'],t[1],t[3]['arg1'])
def p_s3(t):
        '''s : s o s '''
        t[0] = oper(t[2],t[1],t[3])

def p_s4(t):
        '''s : LKEY s RKEY'''
        t[0] = t[2]


def p_g(t):
        '''g : SIN LPAREN FLOAT COMA FLOAT RPAREN
                | LIN LPAREN FLOAT COMA FLOAT RPAREN
                | SIL paren
                | NOI LPAREN FLOAT RPAREN
                | SUB FLOAT %prec UMINUS
                | FLOAT'''
        if t[1] == 'sin':
                t[0] = sin(t[3],t[5])
        elif t[1] == 'lin':
                t[0] = lin(t[3],t[5])
        elif t[1] == 'sil':
                t[0] = sil()
```

```
        elif t[1] == 'noi':
                t[0] = noi(t[3])
        elif t[1] == '-':
                t[0] = [-t[2]]
        elif t[1] == 'sub':
                print 'Error, sub no puede usarse para indicar un numero negativo'
                exit()
        else:
                t[0] = [t[1]]


def p_o(t):
        '''o : CON
                | MIX
                | ADD
                | SUB
                | MUL
                | DIV'''
        t[0] = t[1]

def p_p(t):
        '''p : PLAY LPAREN FLOAT RPAREN
                | POST paren
                | LOOP LPAREN FLOAT RPAREN
                | TUNE LPAREN sign FLOAT RPAREN
                | FILL LPAREN FLOAT RPAREN
                | REDUCE LPAREN FLOAT RPAREN
                | EXPAND paramopcional'''
        if len(t) == 3: #Casos POST y EXPAND
                obj = {'method': t[1], 'arg1':t[2]}
                t[0] = obj
        elif len(t) == 6: #CASO TUNE
                obj = {'method': t[1], 'arg1':t[3]*t[4]}
                t[0] = obj
        else: #Casos PLAY, LOOP, FILL, REDUCE
                obj = {'method': t[1], 'arg1':t[3]}
                t[0] = obj

def p_sign(t):
        '''sign : ADD
                        | SUB
                        | '''
        if len(t) == 2 and t[1] == '-':
                t[0] = -1
        elif len(t) == 2 and t[1] == 'sub':
                print 'Error, sub no puede usarse para indicar un numero negativo'
                exit()
        elif len(t) == 2 and t[1] == 'add':
                print 'Error, add no puede usarse para indicar un numero positivo'
                exit()
        else:
                t[0] = 1


def p_paren(t):
        '''paren : LPAREN RPAREN
                        | '''
        t[0] = None

def p_paramopcional(t):
        '''paramopcional : LPAREN FLOAT RPAREN
                                                | '''
        if len(t) == 3:
```

```
            t[0] = t[2]
        else:
            t[0] = 1

def p_error(t):
        print t
        print("Syntax error at '%s'" % t.value)
```

2.3. Funciones

```
import math,numpy,sys,pygame

global beat
beat = 12

global sample_rate
sample_rate = 8000

#EXCEPCIONES
class NegativeException(Exception):
        @classmethod
        def check(self,x,funcion):
                if x<=0:
                        raise self('Es menor o igual a 0 en la funcion: ',funcion)

#LECTURA DE ARCHIVOS
def leerArchivo(ruta):
        cadena = ""
        fo = open(ruta)
        for line in fo:
                limpia = line
                if ( limpia.find('//') >= 0 ):
                        limpia = limpia[0:limpia.find('//')]
                limpia = limpia.strip().strip('/t')
                cadena+=limpia
        fo.close()
        return cadena


#GENERADORES
def sin(c,a):
        this_function_name = sys._getframe().f_code.co_name

        NegativeException.check(c,this_function_name)
        NegativeException.check(a,this_function_name)

        buff = [0]*beat
        x = (c*2*(math.pi))/beat
        for i in range(0,beat):
                buff[i] = a* (math.sin(i*x))
        return buff

def lin(a,b):
        this_function_name = sys._getframe().f_code.co_name

        NegativeException.check(a,this_function_name)
        NegativeException.check(b,this_function_name)


        return numpy.linspace(a,b,beat)

def sil():
        return [0]*beat
```

```
def noi(a):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(a,this_function_name)

        return numpy.random.random_sample(size=beat)*a



#METODOS
def play(buff,ms):
        pygame.mixer.pre_init(sample_rate, -16, 1) # 44.1kHz, 16-bit signed, mono
        pygame.init()

        buffO = buff

        buff = numpy.array(buff)

        ms = int(round(ms))

        sound = pygame.sndarray.make_sound(buff)
        sound.play(-1)
        pygame.time.delay(ms)
        sound.stop()

        return buffO

def post(buff,p=None):
        #Aca no importa que sea p
        cadena = ""
        for i in range(0,len(buff)):
                cadena= cadena+" " +str(buff[i])
        print cadena

        return buff

def loop(buff,R):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(R,this_function_name)

        buff_r = []
        while R > 0:
                if R < 1:
                        lastItemToAdd = int(round(R*len(buff)))
                        buff_r = buff_r + buff[:lastItemToAdd]
                        R = 0
                else:
                        buff_r = buff_r + buff
                        R -= 1
        return buff_r

def resample(buff_a,L):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(L,this_function_name)

        L = int(round(L))

        buff_b =[0]*L
        for i in range(0,L):
                buff_b[i] = buff_a[i*len(buff_a)//L]
        return buff_b

def tune(buff , P):
```

```
        #Acepta negativos, 0 y positivos
        return resample(buff,int(len(buff)*((2**(1.0/beat))**-P)))

def reduce(buff, N):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(N,this_function_name)

        L = beat*N
        if len(buff)>N:
                return resample ( buff , N)
        else:
                return buff

def expand(buff, N):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(N,this_function_name)

        L = beat*N
        if len(buff)<N:
                return resample ( buff , N)
        else:
                return buff

def fill(buff,N):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(N,this_function_name)

        L=beat*int(round(N))

        buff_b = [0]*L
        for i in range(0,L):
                if i < len(buff):
                        buff_b[i]=buff[i]
                else:
                        buff_b[i] = 0.0
        return buff_b

def resize(buff_a,L):
        this_function_name = sys._getframe().f_code.co_name
        NegativeException.check(L,this_function_name)

        buff_b = [0]*int(round(L))
        for i in range(0,L):
                buff_b[i] = buff_a[i %len(buff_a)]
        return buff_b


#GENERALIZADORES
def method(op, buff, p):
        return globals()[op](buff,p)

def oper(op, buff_a, buff_b):
        if op == ';' or op == 'con':
                return buff_a+buff_b

        op = function[op]

        if len(buff_a) < len(buff_b):
                a = resize(buff_a, len(buff_b))
                b = buff_b
        else:
                a = buff_a
                b = resize(buff_b, len(buff_a))
```

```
        buff = [0]*len(a)

        for i in range(0,len(a)):
                buff[i] = op(a[i], b[i])
        return buff

def calcularGA(g,a):
        if g != None and a !=None: #Caso Generador y Resto Operador
                while a != None:
                        if len(a) == 3: #Caso methodo
                                g = method(a['method'],g,a['arg1'])
                                a = a['rest']
                        elif len(a) == 2: #Caso operador
                                g = oper(a['operator'],g,a['value'])
                                a = None
                        else: #Caso Vacio
                                a = None
                return g
        else:#Caso Generador y NO Operador
                return g

#OPERADORES
def add(a,b):
        return a + b
def sub(a,b):
        return a-b
def mul(a,b):
        return a*b
def div(a,b):
        return a/b
def mix(a,b):
        return (a+b)/2

function = {'add':add, '+': add,'sub':sub, '-': sub, 'mul':mul, '*': mul,
         'div':div, '/':div, 'mix':mix, '&':mix}
```

2.4. Main

```
from parser import *

import ply.yacc as yacc
yacc.yacc()

def parsearArchivo(ruta):
        cadena = leerArchivo(ruta)
        yacc.parse(linea)

def parsearCadena(cadena):
        yacc.parse(cadena)

while 1:
    try:
         s = input('calc > ')   # Use raw_input on Python 2

    except EOFError:
        break
    yacc.parse(s)
    break
```