Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico 2

Compositor Musical

Teoría de Lenguajes

Primer Cuatrimestre de 2015

Grupo: Autores del Autómata Automático Autodestructivo

Apellido y Nombre	LU	E-mail
Matayoshi, Leandro	79/11	leandro.matayoshi@gmail.com
Panarello, Bernabé	194/01	bpanarello@gmail.com
Vega, Leandro	698/11	leandrogvega@gmail.com

Índice

1. Introducción del problema a resolver

El objetivo de nuestro tp es, dado un archivo de entrada, parsearlo para poder tener un archivo de salida, respetando restricciones solicitadas para el correcto funcionamiento.

1.1. Paso a paso introductivo de la resolución

- Nos dan un lenguaje para descripción de partituras www.dc.uba.ar/materias/tl/2015/c1/tp2-enunciado-compos at_download/file.
- Definimos una gramática para el lenguaje.
- Utilizamos la herramienta PLY configurada con nuestra gramática para generar un AST (árbol sintáctico)
- En base al árbol generado y validado, escribiremos un archivo de salida con el formato especificado en www.dc.uba.ar/materias/tl/2015/c1/tp2-enunciado-compositor-musical/at_download/file.

2. Descripción del problema resuelto

En esta sección explicaremos cada parte implementada para realizar los procedimientos requeridos. Contaremos dudas, errores que fueron surgiendo y explicaremos las decisiones tomadas. Para eso vamos a dividirlo en cuatro secciones que detallamos a continuación ordenadas de las formas en las que lo fuimos realizando.

2.1. Gramática

No contamos con demasiadas dificultades, miramos cada paso de la descripción de la partitura y fuimos creando las producciones necesarias. Se explicita en la sección Gramática.

2.2. PLY

En nuestra implementación utilizamos las herramientas brindadas por PLY como explicamos en la introducción, estas son:

2.2.1. Lexer

Vamos a utilizar un lexer para poder tener definidas nuestras expresiones regulares y poder decidir que cadenas son o no válidas según nuestra gramática. Para generar un lexer vamos a realizar lo descripto en el siguiente link www.dc.uba.ar/materias/tl/2015/c1/files/tp2-clase-intro-a-ply/at_download/file. Crearemos un archivo lexer_rules.py, definiendo los tokens y las reglas. La lista de tokens será explicitada en la sección de la gramática.
Luego pasamos a definir las reglas para cada expresión regular que deseamos tener. En esta parte tuvimos bastantes problemas, principalmente porque optamos por usar reglas 'simples' en todas las reglas.

El primero de los problemas fue al hacer la siguiente expresión regular "$(do|re|mi|fa|sol|la|si)(+|-)?$". Al tenerla toda junta, la regla matcheaba con todas las notas hasta el final del primer paréntesis sin mirar lo que continuaba y, en caso de tener en el archivo de entrada una nota con "$sol+$", buscaba una regla que inicialice con un + o −, la cual no existe, y por lo tanto producía un error. La solución fue separarlas, pudiendo solucionar el problema mencionado.

El segundo problema que surgió fue el tema del orden, al tener reglas como const que generan todo el alfabeto, nos matcheaba voz, compas, entre otras, cuando nosotros en realidad queriamos que esas palabras matcheen en otra regla definida. Como nosotros teníamos definido reglas 'simples', al tratar de ordenar las reglas notamos que la lógica de la clase re (regular expression) tomaba como primera a la que definía en su regla el string más largo, esto nos hizo rever la forma de definir las reglas. Mirando y testeando la clase re, pudimos corroborar que, definiendo las reglas como funciones, se respetaba el orden en las que eran definidas en el script. De esta manera pasamos todas las reglas a funciones como se explica en el pdf del link para reglas 'complejas', logrando salvar dicho problema.

Aún habiendo solucionado los problemas anteriores, surgió un tercer problema. En este caso, se desató un problema con las palabras reservadas. Dado que establecimos un orden previamente, habíamos optado por dejar a la regla *cname* (que es nuestro constructor de constantes) en el último lugar, y que reglas que usaban palabras como *re*, *silencio*, entre otras, se lograra matchear más arriba. Esto produjo que, al definir constantes con el nombre $fantastico$, y al tener reservada la palabra fa, nos imposibilitara usarla produciendo un error. Investigando un poco sobre PLY, logramos encontrar el siguiente enlace http://www.dabeaz.com/ply/ply.html, el cual habla del problema mencionado y describe como corregirlo. La solución consiste en no definir las palabras reservadas como reglas separadas, sino todo junto dentro de, en nuestro caso, *cname*. De esta manera, toda palabra que inicialice con un caracter del alfabeto va entrar por la regla *cname*, detectará la palabra que tenga que detectar y derivará el token correspondiente segun nuestra lista de *reserved*. Se puede apreciar fácilmente que en nuestro ejemplo mencionado, $fantastico$ sólo va a ser derivado a un token $CONST$ mientras que fa será derivado a un token $NOTENAME$, solucionando el problema descripto.

2.2.2. Parser

Vamos a utilizar un parser para poder darle una estructura y una semántica a nuestra grámatica. Para generar un parser vamos a realizar lo descripto en el link mencionado en el lexer desde la página 15 en adelante. Creamos un archivo parser_rules.py, en el cual vamos a definir las producciones de la gramática y cómo generar el arbol AST (abstract Syntax Tree). Este archivo usará los tokens de lexer_rules para construir las producciones, diferenciarlas una de las otras y filtrar aquellas que no sean válidas.
Cada producción llamará a su función interna, que estará definida en parserobject.py, formando el árbol AST desde las hojas hasta su raiz. Cada una de ellas creará un objeto nodo y usará atributos sintetizados para poder intercambiar valores de una rama a la otra y poder validar las condiciones especificadas en nuestro lenguaje, en otras palabras le estaremos dando una semántica a nuestras producciones.

3. Gramática

3.1. Gramática deducida

3.1.1. Tupla

$G = (V_t, V_n, P, H)$

3.1.2. Conjunto finito de terminales (V_t)

{ #tempo, #compas, /, ';', ',', =, (,), {, }, '.', +, −, const, voz, compas, repetir, nota, silencio, blanca, negra, redonda, semicorchea, corchea, fusa, semifusa, do, re, mi, fa, sol, la, si }

3.1.3. Conjunto finito de no terminales (V_n)

{ H, TEMPO, COMPASHEADER, CONSTINIT, CONSTLIST, CONST, VOICELIST, VOICE, VOICE-CONTENT, COMPASLOOP, COMPASLIST, COMPAS, NOTELIST, NOTE, SILENCE, VALUE, SHAPE, NUM, CNAME, NOTENAME, ALTER }

3.1.4. Producciones (P)

H → {TEMPO}{COMPASHEADER}{CONSTINIT}{VOICELIST}
H → {TEMPO}{COMPASHEADER}{VOICELIST}

TEMPO → {tempobegin}{shape}{num}

COMPASHEADER → {compasheaderbegin}{num}{slash}{num}

CONSTINIT → {CONSTLIST}

CONSTLIST → {CONST}
CONSTLIST → {CONSTLIST}{CONST}

CONST → {const}{VALUE}{equals}{VALUE}{semicolon}

VOICELIST → {VOICE}
VOICELIST → {VOICELIST}{VOICE}

VOICE → {voicebegin}{leftpar}{VALUE} {rightpar}{leftcurl}{VOICECONTENT}{rightcurl}

VOICECONTENT → {COMPAS}
VOICECONTENT → {COMPASLOOP}
VOICECONTENT → {VOICECONTENT}{COMPAS}
VOICECONTENT → {VOICECONTENT}{COMPASLOOP}

COMPASLOOP → {loopbegin}{leftpar}{VALUE} rightpar}{leftcurl}{COMPASLIST}{rightcurl}

COMPASLIST → {COMPAS}
COMPASLIST → {COMPASLIST}{COMPAS}

COMPAS → {compasbegin}{leftcurl}{NOTELIST}{rightcurl}

NOTELIST → {NOTE}
NOTELIST → {SILENCE}
NOTELIST → {NOTELIST}{NOTE}
NOTELIST → {NOTELIST}{SILENCE}

NOTE → {notebegin}{leftpar}{notename}{alter}{comma}{VALUE} {comma}{shape}{punto}{rightpar}{semicolon}
NOTE → {notebegin}{leftpar}{notename}{alter}{comma}{VALUE} {comma}{shape}{rightpar}{semicolon}
NOTE → {notebegin}{leftpar}{notename}{comma}{VALUE} {comma}{shape}{punto}{rightpar}{semicolon}
NOTE → {notebegin}{leftpar}{notename}{comma}{VALUE} {comma}{shape}{rightpar}{semicolon}

SILENCE → {silencebegin}{leftpar}{shape}{rightpar}{semicolon}
SILENCE → {silencebegin}{leftpar}{shape}{punto}{rightpar}{semicolon}

VALUE → {cname}
VALUE → {num}

NOTA: Los tokens están definidos como minúsculas para diferenciarlos. A su vez los que su pasaje es trivial (ejemplo 'LOOPBEGIN: repetir'), no están ni como terminal ni no terminal, sino que el que si está es su pasaje trival definido como terminal. Los tokens que generen una expresión regular en el que el pasaje no sea trivial (ejemplo 'NOTENAME: $do|re|mi|fa|sol|la|si$'), estarán definidos como no terminales pese a estár en minúscula.

3.2. Tokens

- TEMPOBEGIN: # tempo
- CONST: const
- EQUALS: =
- SEMICOLON: ;
- VOICEBEGIN: voz
- LEFTPAR: (
- RIGHTPAR:)
- LEFTCURL: {
- RIGHTCURL: }
- COMPASHEADERBEGIN: # compas
- COMPASBEGIN: compas
- LOOPBEGIN: repetir
- SLASH: /
- NOTEBEGIN: nota
- SILENCEBEGIN: silencio
- PUNTO: .
- ALTER: $+|-$
- SHAPE: $blanca|negra|redonda|semicorchea|corchea|semifusa|fusa$
- NOTENAME: $do|re|mi|fa|sol|la|si$

- COMMA: ,
- CNAME: $(([a-z]|[A-Z])([0-9]|[a-z]|[A-Z])*)$
- NUM: $([0]|[1-9][0-9]*)$

4. Tests

4.1. Tests con fallas

4.1.1. Test 1: Tiene compaces con distinta duración

```
#tempo negra 30
#compas 3/4

const oct1 = 2;
const oct2 = 6;
const oct3 = 1;

// Instrumentos
const flauta = 51;

voz (flauta)
{
  compas
  {
    nota(do, oct3, blanca.);
    nota(re, oct1, redonda);
  }

  compas
  {
    nota(mi, oct2, blanca);
    nota(la, oct1, negra);
  }
}
```

4.1.2. Test 2: Tiene voces con compaces de distinta duración

```
#tempo negra 120
#compas 2/2

const oct1 = 5;
const oct2 = 2;
const oct3 = 4;

// Instrumentos
const violin = 20;
const guitarra = 12;

voz (violin)
{
  compas
  {
    nota(do, oct3, blanca.);
    nota(re, oct1, negra);
  }

  compas
  {
    nota(mi, oct2, blanca.);
    nota(la, oct1, negra);
  }
}

voz (guitarra)
{
```

```
  compas
  {
    nota(do, oct3, fusa);
    nota(re, oct1, semifusa.);
  }

  compas
  {
    nota(mi, oct2, fusa);
    nota(la, oct1, semifusa.);
  }
}
```

4.1.3. Test 3: Constante que apunta a una constante no definida

```
#tempo negra 60
#compas 1/1

const oct1 = 3;
const oct2 = ConstanteTrucha;

// Instrumentos
const bajo = 20;

voz (bajo)
{
  compas
  {
    nota(do, oct1, blanca.);
    nota(re, oct1, negra);
  }

  compas
  {
    nota(mi, oct2, blanca.);
    nota(la, oct2, negra);
  }
}
```

4.1.4. Test 4: Constante definida circularmente

```
#tempo negra 60
#compas 2/8

const oct1 = 3;
const oct2 = 5;

// Instrumentos
const bajo = 20;
const malPensado = malPensado;

voz (bajo)
{
  compas
  {
    nota(do, oct1, corchea.);
    nota(re, oct1, semicorchea);
  }

  compas
  {
```

```
21      nota(mi, oct2, semicorchea);
22      nota(la, oct2, corchea.);
23    }
24  }
```

4.2. Tests correctos

4.2.1. Test 1: Simple

```
1  #tempo negra 30
2  #compas 2/4
3
4  const oct1 = 2;
5  const oct2 = 6;
6  const oct3 = 1;
7
8  // Instrumentos
9  const flauta = 51;
10
11 voz (flauta)
12 {
13   compas
14   {
15     nota(do, oct3, negra);
16     nota(re, oct1, negra);
17   }
18
19   compas
20   {
21     nota(mi, oct2, blanca);
22   }
23 }
```

4.2.2. Test 2: Con varias voces

```
1  #tempo negra 120
2  #compas 3/4
3
4  const oct1 = 2;
5  const oct2 = 6;
6  const oct3 = 1;
7  const oct4 = 3;
8
9  // Instrumentos
10 const piano = 65;
11 const violin = 31;
12
13 voz (piano)
14 {
15   compas
16   {
17     nota(sol, oct3, blanca);
18     nota(fa+, oct2, negra);
19   }
20
21   compas
22   {
23     nota(mi, oct2, negra.);
24     nota(fa, oct4, corchea);
25     nota(mi, oct2, negra);
```

```
26    }
27    compas
28    {
29        silencio (negra);
30      nota(sol-, oct1, negra);
31      nota(sol-, oct1, negra);
32    }
33  }
34
35  voz ( violin )
36  {
37    compas
38    {
39      nota(la, oct1, blanca.);
40    }
41
42    compas
43    {
44      nota(mi, oct2, negra.);
45      nota(fa, oct4, corchea);
46      nota(mi, oct2, negra);
47    }
48    compas
49    {
50        silencio (semicorchea);
51      nota(mi, oct1, semicorchea);
52      nota(sol, oct4, corchea);
53      nota(sol, oct4, blanca);
54    }
55  }
```

4.2.3. Test 3: Con repeticiones

```
1  #tempo negra 120
2  #compas 2/4
3
4  const oct1 = 2;
5  const oct2 = 6;
6  const oct3 = 1;
7  const oct4 = 3;
8
9  // Instrumentos
10 const flauta = 10;
11 const violin = 31;
12
13 voz (flauta)
14 {
15   repetir (5)
16   {
17     compas
18     {
19         nota(sol, oct3, negra);
20         nota(fa+, oct2, negra);
21     }
22
23     compas
24     {
25         nota(mi, oct2, negra.);
26         nota(fa, oct4, corchea);
27     }
28     compas
29     {
```

```
30          nota(sol, oct1, blanca);
31      }
32    }
33  }
34
35  voz ( violin )
36  {
37    compas
38    {
39      nota(la, oct1, blanca);
40    }
41
42    compas
43    {
44      nota(fa, oct4, corchea);
45      nota(mi, oct2, negra .);
46    }
47    compas
48    {
49       silencio (semifusa);
50      nota(si+, oct1, semifusa);
51      nota(si, oct4, semifusa);
52      nota(fa, oct4, semifusa);
53      nota(sol, oct4, semicorchea);
54      nota(mi-, oct4, corchea);
55      nota(re, oct4, negra);
56    }
57  }
```

5. Manual del programa

5.1. Modo de uso

Línea de ejecución: ./musileng entrada.mus salida.txt

5.1.1. Reglas para evitar posibles errores

Se aceptan archivos de entrada que contengan:

- Todos los compases deben tener la misma duración al sumar la duración de sus notas y/o silencios.
- Todas las voces deben tener la misma cantidad de compases.
- No deben existir constantes indefinidas o definidas circularmente. Ejemplo constante no definida: $const\ eval = hola$; (hola jamás se definió). Ejemplo constante definida circularmente: $const\ eval1 = eval2$; $const\ eval2 = eval1$;.
- La suma de la duración de cada compas debe ser igual a $num1/num2$, donde $num1$ y $num2$ son los números definidos en $\#compas\ num1/num2$ en el encabezado.
- Una constante definida como instrumento sólo acepta valores del 1 al 127.
- El valor colocado en 'repetir' debe ser mayor a 0.
- Una constante definida como octava sólo acepta valores del 1 al 9.
- No debe haber una constante definida más de una vez.
- En $\#compas\ num1/num2$ del encabezado, $num2$ debe ser un número que corresponda al de las figuras definidas, para corroborar puede ver la página 2 de www.dc.uba.ar/materias/tl/2015/c1/tp2-enunciado-compositor-musical/at_download/file en la tabla Figura y Valor numérico asociado.
- Partituras incompletas. Esto es, por ejemplo '$\#tempo\ negra$ 120 $\#compas$ 3/4' y no tenér más nada en el archivo de entrada. Se puede observar que incluye el caso de un archivo vacío.
- El número del tempo asignado en el encabezado debe ser mayor a 0.

En caso contrario que no se respete lo mencionado anteriormente, nuestro programa especificará el error cometido para que pueda solucionarlo.

Para más información sobre los archivos de entrada y salida puede mirarse el siguiente pdf www.dc.uba.ar/materias/tl/2015/c1/tp2-enunciado-compositor-musical/at_download/file.

5.2. Requerimientos necesarios para ejecutar

- Programa: Python
- Versión: 2.7

6. Conclusiones

- PLY es una herramienta muy útil, facilita el parseo y nos permite, de manera mucho más corta y sencilla, realizar las diferentes tareas para nuestro lenguaje.
- Lo más complicado fue la parte del lexer, específicamente en la parte que definimos los tokens. Porque como explicamos en la descripción necesitabamos establecer un orden y para eso tuvimos que, entre otras cosas, entender como funcionaba y probar la clase re (regular expression) de python.
- Los temas aportados en las clases, como gramática de atributos, TDS, parser, gramática LALR fueron útiles para poder resolver los problemas presentados.

7. Apéndices

7.1. musileng.py

```
#!/usr/bin/env python
from parserobjects import *


#Programa principal MUSILEN

import lexer_rules
import parser_rules

from sys import argv, exit
from midi import *
from ply.lex import lex
from ply.yacc import yacc


if __name__ == "__main__":
    if len(argv) != 3:
        print ("Invalid arguments.")
        print ("Use:")
        print ("  ./musileng.py <archivo_entrada> <archivo_salida>")
        exit(1)

    try:
        inputfile = argv[1]
        outputfile = argv[2]

        with open (inputfile, "r") as myfile:
            text=myfile.read()

        ConstantManager.createInstance([],[])

        lexer = lex(module=lexer_rules)
        parser = yacc(module=parser_rules)

        expression = parser.parse(text, lexer)
        midi = MidiTranslator()

        with open (outputfile, "w") as file_output:
            midi.generateMIDIFile(expression,file_output)


    except Exception as ex:
        print('ERROR:')
        for v in ex.args:
            print(str(v))

        exit(1)
```

7.2. lexer_rules.py

```
tokens = [
   'COMMENT',
   'NUM',
   'CONST',
   'EQUALS',
   'CNAME',
   'SILENCEBEGIN',
   'SEMICOLON',
   'VOICEBEGIN',
   'LEFTPAR',
   'RIGHTPAR',
   'LEFTCURL',
   'RIGHTCURL',
   'COMPASBEGIN',
   'COMPASHEADERBEGIN',
   'LOOPBEGIN',
   'SLASH',
   'NOTEBEGIN',
   'SHAPE',
   'PUNTO',
   'ALTER',
   'NOTENAME',
   'COMMA',
   'NEWLINE',
   'TEMPOBEGIN'
]

#Ignoro espacios y tabs
t_ignore  = ' \t'

#Ignorar comoentarios
def t_COMMENT(token):
  r'//.*'

def t_NEWLINE(token):
  r"\n+"
  token.lexer.lineno += len(token.value)

#Chequear regexs con: https://regex101.com/#python
# Tener en cuenta poner las expresiones que matchean strings mas largos primero
# cuando hay ambiguedad.
# Usamos '\s' para matchear espacios en blanco
def t_TEMPOBEGIN(token):
  r"\#tempo"
  return token


def t_EQUALS(token):
  r"\="
  return token

def t_SEMICOLON(token):
  r"\;"
  #print ("semicolon")
  return token
```

```
def t_LEFTPAR(token):
  r"\("
  return token

def t_RIGHTPAR(token):
  r"\)"
  return token

def t_LEFTCURL(token):
  r"\{"
  return token

def t_RIGHTCURL(token):
  r"\}"
  return token

def t_COMPASHEADERBEGIN(token):
  r"\#compas"
  return token

def t_SLASH(token):
  r"/"
  return token

def t_PUNTO(token):
  r"\."
  return token

def t_ALTER(token):
  r"\+|\-"
  return token

def t_COMMA(token):
  r"\,"
  return token

def t_CNAME(token):
  r"(([a-z]|[A-Z])([0-9]|[a-z]|[A-Z]|\_)*)"

  if reserved.get(token.value) != None:
      token.type = reserved.get(token.value)

  return token

def t_NUM(token):
    r"([0]|[1-9][0-9]*)"
    token.value = int(token.value)
    return token

def t_error(token):
    message = "[Lexer] Token desconocido"
    message += "\nline:" + str(token.lineno)
    message += "\nposition:" + str(token.lexpos)
    raise Exception(message)
```

```
reserved = {
    'const':'CONST',
    'do' : 'NOTENAME',
    're' : 'NOTENAME',
    'mi' : 'NOTENAME',
    'fa' : 'NOTENAME',
    'sol': 'NOTENAME',
    'la' : 'NOTENAME',
    'si' : 'NOTENAME',
    'nota' : 'NOTEBEGIN',
    'compas' : 'COMPASBEGIN',
    'repetir': 'LOOPBEGIN',
    'voz': 'VOICEBEGIN',
    'negra': 'SHAPE',
    'blanca' :'SHAPE',
    'redonda' : 'SHAPE',
    'corchea' : 'SHAPE',
    'semicorchea' : 'SHAPE',
    'fusa' : 'SHAPE',
    'semifusa' : 'SHAPE',
    'silencio' : 'SILENCEBEGIN',

}
```

7.3. parserobjects.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
__author__ = 'AAAA'

#En este archivo implementamos todas las clases que representan los objetos
#que se van parseando (no terminales),
#algunas clases auxiliares y el manejador de constantes.

#Clase abstracta que representa expresiones (nodos en a gramática)
class Expression(object):
    pass

#Representa una lista como un valor y un puntero a otra lista
#Por ejemplo, A -> id, A -> A id.
# Notar que los elementos de la lista están en orden inverso
class ExpressionList(Expression):
    def __init__(self, current, nextList):
        self._list = nextList + [current]

    def getList(self):
        return self._list

#Representa un nodo del elmento const en la gramática
class Const(Expression):
    def __init__(self, cname, val, isPointer):
        self._cname = cname
        self._value = val
        self._isPointer = isPointer

    def getName(self):
        return self._cname

    def getValue(self):
        return self._value

    #indica si la constante apunta a otra constante
    def isPointer(self):
        return self._isPointer


#Representa una lista de constantes
class ConstList(ExpressionList):
    pass

#Representa la lista de contenido de una voz (compaces o loops)
class VoiceContent(ExpressionList):
    def __init__(self, current, nextList):
        if nextList != [] and current.getDuration() != nextList[0].getDuration():
            raise Exception("Se detectó un compas dentro de una voz con
            duración distinta al resto de los compases dentro de la misma.")

        self._duration = current.getDuration()
        super(VoiceContent,self).__init__(current, nextList)

    def getDuration(self):
        return self._duration
```

```
#Representa una lista de constantes dentro de un loop
class CompasList(ExpressionList):
    def __init__(self, current, nextList):
        if nextList != [] and current.getDuration() != nextList[0].getDuration():
            raise Exception("Se detectaron compases con distinta duración
            dentro de un bucle.")

        self._duration = current.getDuration()
        super(CompasList,self).__init__(current, nextList)

    def getDuration(self):
        return self._duration

#Manejador de constantes. Mantiene una lista de todas las constantes
#declaradas y permite obtener
#los valores de las mismas dado un nombre de constante.
class ConstantManager:
    #Inicializa el manejador de constantes dada una lista de objetos tipo Const.

    @staticmethod
    def createInstance(constList, reserved):
        ConstantManager._instance = ConstantManager(constList, reserved)

    @staticmethod
    def getInstance():
        if ConstantManager._instance == None:
            ConstantManager.createInstance([],[])
        return ConstantManager._instance


    #Inicialización y validación de consistencia de la declaración de constantes
    def __init__(self, constList, reserved):
        self.dictConst = {}
        for c in constList:
            if reserved.count(c.getName()) > 0:
                raise Exception("El nombre de constante <{0}> es una
                palabra reservada".format(c.getName()))

            self.Add (c.getName(), c)

        #Pido los valores de todas las ctes para que se validen
        for c in constList:
            self.getValue(c.getName())

    def Add(self, cname, const):
        if self.dictConst.get(cname) != None:
            raise Exception("Constante duplicada: {0}".format(cname))

        self.dictConst[cname] = const

    #Devuelve el valor de una constante. Si es una referencia a otra
    #constante, calcula primero el valor recursivamente
    def getValue(self, cname):
        const = self.dictConst.get(cname)
        if const == None:
            raise Exception("Constante no definida: {0}".format(cname))
        else:
```

```
            constVal = self.resolveVal(const)

        return constVal


    #Calcula el valor de una constante. Si es numérica (caso base),
    #devuelve su valor.
    #Si es un puntero, busca el valor de la constante a la que apunta
    #(recursivamente), chequeando que no hayan loops.
    def resolveVal(self, const):
        if (not const._isPointer):
            return const.getValue()

        found = False
        visited = [const.getName()]
        next = const
        while(not found):
            if (next.isPointer()):
                nextcname = next.getValue()
                if visited.count(nextcname) > 0:
                    raise Exception("Definición circular de
                    constante <{0}>".format(const.getName()))

                next = self.dictConst.get(nextcname)
                if (next == None):
                    raise Exception("La constante <{0}> apunta
                    (directa o indirectamente) al nombre <{1}> que no
                    está definido".format(const.getName(),nextcname))
                visited.append(nextcname)
            else:
                #const.setValue(next.getValue())
                found = True

        return next.getValue()



class Tempo(Expression):
    def __init__(self, shape, num):
        if num<=0:
            raise Exception("El tempo debe ser mayor a 0")

        self._shape = shape
        self._num = num

    def getFigure(self):
        return self._shape

    def getCount(self):
        return self._num

    def getShapeDuration(self):
        return duration_from_shape(self.getFigure())

#Representa el nodo que define la duración de los compaces para toda la partitura
class CompasHeader(Expression):
    def __init__(self, num1, num2):
        if num2 == 0:
```

```
            raise Exception("El denominador del compas debe ser mayor a cero")
        if not num2 in (1,2,4,8,16,32,64):
            raise Exception("El denominador definido para el tempo de
            los compaces no corresponde a una figura válida.")


        self._numerator = num1
        self._denominator = num2


    #Devuelve la duración del compas, en redondas
    def getDuration(self):
        return float(self._numerator) / float(self._denominator)


    def getDenominator(self):
        return self._denominator


    def getNumerator(self):
        return self._numerator

#Representa un nodo voz(N)
class Voice(Expression):
    def __init__(self, value, voiceContent):
        if value > 127:
          raise Exception("El instrumento {0} es inválido".format(value))

        self._value = value
        self._compasses = voiceContent.getList()

    def getInstrument(self):
        return self._value

    def getCompasses(self):
        return self._compasses


#Representa un nodo repetir(N) con una lista de compaces adentro
class CompasLoop(Expression):
    def __init__(self, value, compasList):
        if value < 1:
            raise Exception("Existe un bucle con cantidad de repeticiones 0.
            Los bucles deben tener al menos una repeticion.")

        self._compasList = compasList
        self._value = value
        self._duration = compasList.getDuration()


    def getRepeat(self):
        return self._value

    def getCompasses(self):
        return self._compasList

    def getDuration(self):
        return self._duration
```

```
    def isLoop(self):
        return True


def duration_from_shape(shape):
    if shape == 'redonda':
        duration = 1.0
    elif shape == 'blanca':
        duration = 1.0 / 2.0
    elif shape == 'negra':
        duration = 1.0 / 4.0
    elif shape == 'corchea':
        duration = 1.0 / 8.0
    elif shape == 'semicorchea':
        duration = 1.0 / 16.0
    elif shape == 'fusa':
        duration = 1.0 / 32.0
    elif shape == 'semifusa':
        duration = 1.0 / 64.0
    else:
        raise Exception("Figura no definida")
    return duration


class Silence(Expression):
    def __init__(self, duration, puntillo):
        self._duration = duration_from_shape(duration)
        if puntillo:
            self._duration = self._duration * (3.0 / 2.0)

    def getDuration(self):
        return self._duration

    def getShape(self):
        return self._shape

    def isSilence(self):
        return True


class Note(Expression):
    def __init__(self, notename, alter, value, shape, puntillo):
        if value < 1 or value > 9:
            raise Exception("La octava {0} es inválida".format(value))

        # concatenacion de string
        if alter != None:
            self._height = notename + alter
        else:
            self._height = notename

        self._value = value
        self._duration = duration_from_shape(shape)

        if puntillo:
            self._duration = self._duration * (3.0 / 2.0)
```

```
    def getOctave(self):
        return self._value

    def getHeight(self):
        return self._height

    def getDuration(self):
        return self._duration

    def isSilence(self):
        return False

#Objeto nodo principal
class Root(Expression):
    def __init__(self, tempo, compasheader,constlist,voicelist):
        ### El tiempo del primer compas de la primera voz debe coincidir
        ###con el compasheader
        if voicelist.getList()[0].getCompasses()[0].getDuration()
        != compasheader.getDuration():
          raise Exception("La duracion de los compases es distinta a la
          del encabezado")
        self._tempo = tempo
        self._compasheader = compasheader
        self._constlist = constlist
        self._voicelist = voicelist

    def getDuration(self):
        return self._tempo

    def getTempo(self):
        return self._tempo

    def getCompasHeader(self):
        return self._compasheader

    def getConstList(self):
        return self._constlist

    def getVoiceList(self):
        return self._voicelist

class VoiceList(Expression):
    def __init__(self, voice, voicelist=[]):
        if len(voicelist)>0:
            ### Deberíamos reescribir esto. Encapsular mediante funciones
            #if (len(voice.getCompasses() !=
            # len(voicelist.first.getCompasses()):
            #    raise Exception("Las voces tienen distinta cantidad
            #de compases")
            ### Un compás de la nueva voz tiene que durar lo mismo que un
            ### compás de la lista
            if (voice.getCompasses()[0].getDuration() !=
             voicelist[0].getCompasses()[0].getDuration()):
                raise Exception("La duracion de los compases de las voces
                son distintas")
            if len(voicelist)>=16:
             raise Exception("La cantidad de voces supera las 16")
```

```
        self._voicelist = voicelist + [voice]

    def getList(self):
        return self._voicelist

class Compas(Expression):
    def __init__(self, notelist):
        self._notelist = notelist
        ### Calculo la duracion del compas
        duration = 0.0
        for note in notelist:
            duration += note.getDuration()
        self._duration = duration

    def getDuration(self):
      return self._duration

    def isLoop(self):
        return False

    def getNoteList(self):
        return self._notelist

class NoteList(Expression):
    def __init__(self, note, notelist):
        self._notelist = notelist + [note]

    def getNoteList(self):
      return self._notelist
```

7.4. parser_rules.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
from parserobjects import *
from lexer_rules import tokens

def p_root(subexpressions):
    'h : tempo compasheader constlistinit voicelist'
    subexpressions[0] = Root(subexpressions[1], subexpressions[2],
     subexpressions[3], subexpressions[4])

def p_root_no_const(subexpressions):
    'h : tempo compasheader voicelist'
    subexpressions[0] = Root(subexpressions[1], subexpressions[2], None,
     subexpressions[3])

def p_tempo(subexpression):
    'tempo : TEMPOBEGIN SHAPE NUM'
    subexpression[0] = Tempo(subexpression[2], int(subexpression[3]))

def p_compasheader(subexpression):
    'compasheader : COMPASHEADERBEGIN NUM SLASH NUM'
    subexpression[0] = CompasHeader(int(subexpression[2]), int(subexpression[4]))

def p_voice(subexpression):
    'voice : VOICEBEGIN LEFTPAR value RIGHTPAR LEFTCURL voicecontent RIGHTCURL'
    subexpression[0] = Voice(subexpression[3], subexpression[6])

def p_compasloop(subexpression):
    'compasloop : LOOPBEGIN LEFTPAR value RIGHTPAR LEFTCURL compaslist
     RIGHTCURL'
    subexpression[0] = CompasLoop(subexpression[3], subexpression[6])

def p_note(subexpression):
    'note : NOTEBEGIN LEFTPAR NOTENAME COMMA value COMMA SHAPE RIGHTPAR
     SEMICOLON'
    subexpression[0] = Note(subexpression[3], None, subexpression[5],
     subexpression[7], False)

def p_note_alter(subexpression):
    'note : NOTEBEGIN LEFTPAR NOTENAME ALTER COMMA value COMMA SHAPE RIGHTPAR
     SEMICOLON'
    subexpression[0] = Note(subexpression[3], subexpression[4],
     subexpression[6], subexpression[8], False)

def p_note_punto(subexpression):
    'note : NOTEBEGIN LEFTPAR NOTENAME COMMA value COMMA SHAPE PUNTO RIGHTPAR
     SEMICOLON'
    subexpression[0] = Note(subexpression[3], None, subexpression[5],
     subexpression[7], True)

def p_note_alter_punto(subexpression):
    'note : NOTEBEGIN LEFTPAR NOTENAME ALTER COMMA value COMMA SHAPE PUNTO
     RIGHTPAR SEMICOLON'
    subexpression[0] = Note(subexpression[3], subexpression[4],
     subexpression[6], subexpression[8], True)
```

```
def p_compaslist_base(subexpression):
    'compaslist : compas'
    subexpression[0] = CompasList(subexpression[1], [])

def p_compaslist_rec(subexpression):
    'compaslist : compaslist compas'
    subexpression[0] = CompasList(subexpression[2], subexpression[1].getList())

def p_voice_list_base(subexpression):
    'voicelist : voice'
    subexpression[0] = VoiceList(subexpression[1])

def p_voice_list_rec(subexpressions):
    'voicelist : voicelist voice'
    ### Invierto parametros intencionalmente. voicelist param es opcional en
    ### el new de la clase
    subexpressions[0] = VoiceList(subexpressions[2], subexpressions[1].getList())

def p_compas(subexpressions):
    'compas : COMPASBEGIN LEFTCURL notelist RIGHTCURL'
    subexpressions[0] = Compas(subexpressions[3].getNoteList())

def p_note_list_base_note(subexpression):
    'notelist : note'
    subexpression[0] = NoteList(subexpression[1], [])

def p_note_list_base_silence(subexpression):
    'notelist : silence'
    subexpression[0] = NoteList(subexpression[1], [])

def p_note_list_rec_note(subexpressions):
    'notelist : notelist note'
    ### Invierto parametros intencionalmente. notelist param es opcional en
    ### el new de la clase
    subexpressions[0] = NoteList(subexpressions[2], subexpressions[1].getNoteList())

def p_note_list_rec_silence(subexpressions):
    'notelist : notelist silence'
    ### Invierto parametros intencionalmente. notelist param es opcional en
    ### el new de la clase
    subexpressions[0] = NoteList(subexpressions[2], subexpressions[1].getNoteList())

def p_val_num(subexpression):
    'value : NUM'
    subexpression[0] = int(subexpression[1])


def p_val_cname(subexpression):
    'value : CNAME'
    subexpression[0] = ConstantManager.getInstance().getValue(subexpression[1])
def p_const(subexpressions):
    'const : CONST CNAME EQUALS NUM SEMICOLON'
    subexpressions[0] = Const(subexpressions[2],int(subexpressions[4]), False)

#Una constante que es un puntero a otra constante
def p_const_cname(subexpressions):
    'const : CONST CNAME EQUALS CNAME SEMICOLON'
    subexpressions[0] = Const(subexpressions[2],subexpressions[4], True)
```

```
#Es un pasamanos para poder inicializar el ConstantManager y que pueda ser
#usado por las otras producciones
#(esto asume que las constantes se declaran primero en el header)
def p_const_list_init(subexpressions):
    'constlistinit : constlist'
    subexpressions[0] = subexpressions[1]
    #Sabemos que subexpressions[0] es un constlist, inicializamos el
    # Constantmanager para que el resto
    #de las producciones puedan referenciar constantes.
    #TODO: Pasarle el reserved del Lexer
    ConstantManager.createInstance (subexpressions[0].getList(),[] )


def p_const_list_base(subexpressions):
    'constlist : const'
    subexpressions[0] = ConstList(subexpressions[1],[])

def p_const_list_rec(subexpressions):
    'constlist : constlist const'
    subexpressions[0] = ConstList(subexpressions[2],subexpressions[1].getList())


def p_voice_content_base_loop(subexpressions):
    'voicecontent : compasloop'
    subexpressions[0] = VoiceContent(subexpressions[1],[])

def p_voice_content_base_compas(subexpressions):
    'voicecontent : compas'
    subexpressions[0] = VoiceContent(subexpressions[1],[])

def p_voice_content_rec_compasloop(subexpressions):
    'voicecontent : voicecontent compasloop'
    subexpressions[0] = VoiceContent(subexpressions[2],subexpressions[1].getList())

def p_voice_content_rec_compas(subexpressions):
    'voicecontent : voicecontent compas'
    subexpressions[0] = VoiceContent(subexpressions[2],subexpressions[1].getList())

def p_silence(subexpression):
    'silence : SILENCEBEGIN LEFTPAR SHAPE RIGHTPAR SEMICOLON'
    subexpression[0] = Silence(subexpression[3],None)

def p_silence_punto(subexpression):
    'silence : SILENCEBEGIN LEFTPAR SHAPE PUNTO RIGHTPAR SEMICOLON'
    subexpression[0] = Silence(subexpression[3],True)


def isReserved(token):
    return token in ('const',
    'do',
    're',
    'mi',
    'fa',
    'sol',
    'la',
    'si',
    'tempo',
```

```
    'compas',
    'repetir',
    'voz',
    'negra',
    'blanca',
    'redonda',
    'corchea',
    'smicorchea',
    'fusa',
    'semifusa')

def p_error(subexpressions):
    #print ("----------------------------")
    #print ("----------------------------")
    #print (subexpressions)

    if (subexpressions != None):
        if isReserved(subexpressions.value):
            strReservedMsg = '(palabra reservada)'
        else:
            strReservedMsg = ''

        raise Exception("[Parser] Error de sintaxis Linea: {0}, Pos (absoluta):
         {1}, Token: <{2}>{3} ".format(subexpressions.lineno,
          subexpressions.lexpos, subexpressions.value, strReservedMsg))
    else:
        raise Exception("[Parser] Archivo incompleto")
```

7.5. midi.py

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
from parserobjects import *

class MidiTranslator:
    def generateMIDIFile(self, root, file_output):

        strFormatHeader0 = "MFile 1 {0} 384"
        strFormatHeader1 = "MTrk"
        strFormatHeader2 = "000:00:000 Tempo {0}"
        strFormatHeader3 = "000:00:000 TimeSig {0}/{1} 24 8"
        strFormatHeader4 = "000:00:000 Meta TrkEnd"
        strFormatHeader5 = "TrkEnd"


        strFormatVoyHeader1 = 'MTrk'
        strFormatVoyHeader2 = '000:00:000 Meta TrkName "Voz {0}"'
        strFormatVoyHeader3 = '000:00:000 ProgCh ch={0} prog={1}'


        strFormatVoyHeader4 = '{0:03d}:{1:02d}:{2:03d} Meta TrkEnd'
        strFormatVoyHeader5 = "TrkEnd"

        self.writeMidi(strFormatHeader0.format(len(root.getVoiceList().getList())
                       + 1),file_output)
        self.writeMidi(strFormatHeader1,file_output)
        self.writeMidi(strFormatHeader2.format(self.calculateTempo(
                       root.getTempo())),file_output)
        self.writeMidi(strFormatHeader3.format(root.getCompasHeader().getNumerator(),
                       root.getCompasHeader().getDenominator()),file_output)
        self.writeMidi(strFormatHeader4,file_output)
        self.writeMidi(strFormatHeader5,file_output)

        vlist = root.getVoiceList().getList()

        for i in range(0, len(vlist)):
            #Encabezado de la Voz
            v = vlist[i]
            channel = i + 1
            self.writeMidi(strFormatVoyHeader1,file_output)
            self.writeMidi(strFormatVoyHeader2.format(channel),file_output)
            self.writeMidi(strFormatVoyHeader3.format(channel,
                           v.getInstrument()),file_output)
            compasId = 0
            vc = v.getCompasses()
            for j in range(0, len(vc)):
                content = vc[j]

                if content.isLoop():
                    repeat = content.getRepeat()
                    contentList = content.getCompasses().getList()
                else:
                    repeat = 1
                    contentList = [content]

                for k in range(0, repeat):
```

```
                for c in contentList:
                    self.writeCompass(c, root, compasId, channel,
                                      file_output)
                    compasId = compasId + 1

        self.writeMidi(strFormatVoyHeader4.format(compasId, 0, 0)
                       ,file_output)

        self.writeMidi(strFormatVoyHeader5,file_output)



def writeMidi(self, line, file_output):
    file_output.writelines(line)
    file_output.write('\n')

def writeCompass(self, compass, root, compassId, channel, file_output):

    strFormatVoyNoteOn = '{0:03d}:{1:02d}:{2:03d} On ch={3} note={4}{5}
                         vol=70'
    strFormatVoyNoteOff = '{0:03d}:{1:02d}:{2:03d} Off ch={3} note={4}{5}
                          vol=0'

    clickOffset = 0
    pulseOffset = 0
    nList = compass.getNoteList()
    for i in range (0, len(nList)):
        n = nList[i]
        clicks = int(n.getDuration() *
                     root.getCompasHeader().getDenominator() * 384)
        finalClick = (clicks + clickOffset) % 384

        finalPulse = pulseOffset + (clicks + clickOffset) // 384


        if not n.isSilence():
            noteName = self.fromLatinToAmerican(n.getHeight())
            octave = n.getOctave()
            self.writeMidi(strFormatVoyNoteOn.format(compassId, pulseOffset,
                           clickOffset, channel, noteName, octave),
                           file_output)

            #Imprimo la nota en el archivo. Si la nota es la ultima del
            #compas, entonces imprimo el Off al comienzo del proximo compas
            if i < len(nList) - 1:

                self.writeMidi(strFormatVoyNoteOff.format(compassId,
                               finalPulse, finalClick, channel, noteName,
                               octave),file_output)
            else:
                self.writeMidi(strFormatVoyNoteOff.format(compassId + 1, 0,
                               0, channel, noteName, octave),file_output)


        clickOffset = finalClick
        pulseOffset = finalPulse
```

```
def calculateTempo(self, tempo):
    return int(1000000 * 60 / (tempo.getShapeDuration() * 4 * tempo.getCount()))

def fromLatinToAmerican(self, noteName):
    #A: la
    #B: si
    #C: do
    #D: re
    #E: mi
    #F: fa
    #G: sol.

    return noteName.replace('la', 'a').replace('si','b').replace('do',
           'c').replace('re', 'd').replace('mi', 'e').replace('fa',
           'f').replace('sol', "g")
```