DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP2

8/7/2015 Teoría de Lenguajes

PLD

Integrante	LU	Correo electrónico
Pablo Herrero	332/07	pablodherrero@gmail.com
Leandro Tozzi	-	leandro.tozzi@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

Contents

1 Introducción

En este trabajo desarrollamos un parser para el lenguaje musileng, creado por el cuerpo docente de la materia Teoría de Lenguajes de nuestra facultad. Este lenguaje está orientado a la composición de piezas musicales, generando un archivo en un lenguaje intermedio, que luego será transformado al formato MIDI para su reproducción.

2 Especificación de la gramática

A continuación detallamos primero los tokens reconocidos por el lexer y luego las producciones reconocidas por el parser.

2.1 Tokens

En la siguiente tabla podemos observar los tokens reconocidos por el Lexer. Las cadenas representadas por cada token se corresponden con su nombre en minúscula salvo especificado lo contrario.

Token	Descripción
HASH	Carácter "#"
EQUALS	Carácter "="
SLASH	Carácter "/"
ID	Constante alfanumérica (expresión regular `[_a-zA-Z][_a-zA-Z0-9]*`)
MUSICAL_NOTE	Constante alfanumérica que representa las notas musicales: `do re mi fa sol la si`
NOTE_VALUE	Constantes que representan los valores de las notas musicales: ``redonda'', ``blanca'', ``negra'', ``corchea'', ``semicorchea'', ``fusa'', ``semifusa''
SEMICOLON	Carácter ";"
COMMA	Carácter ","
POINT	Carácter "."
NUMBER	Constante numérica (expresión regular `[0-9]+(\.[0-9]+)?`)
PLUS, MINUS	Operadores que modifican una nota musical (caracteres "+", "-")
LPAREN, RPAREN	Agrupación (caracteres "(" y ")")
LBRACKET, RBRACKET	Delimitación de bloques "{" y "}")
COMMENT	Comentario (expresión regular `"([^\\\n]|(\\.))*"`)
CONST	Cadena "const"
COMPAS	Cadena "compas"
TEMPO	Cadena "tempo"
VOZ	Cadena "voz"
REPETIR	Cadena "repetir"
NOTA	Cadena "nota"
SILENCIO	Cadena "silencio"

2.2 Producciones

Las producciones que definen la gramática se detallan a continuación, en el formato en el que son requeridas por la librería PLY.
La producción inicial de esta gramática es `musileng`.

```
musileng         : tempo_directive compas_directive constants voices

tempo_directive  : HASH TEMPO NOTE_VALUE NUMBER

compas_directive : HASH COMPAS NUMBER SLASH NUMBER

constants        : constant constants
                 |

constant         : CONST ID EQUALS NUMBER SEMICOLON

voices           : voice voices
                 | voice

voice            : VOZ LPAREN numeric_value RPAREN LBRACKET voice_content RBRACKET
```

```
  voice_content    : compas voice_content
                   | repetition voice_content
                   | compas
                   | repetition

  compas           : COMPAS LBRACKET compas_content RBRACKET

  compas_content   : note compas_content
                   | silence compas_content
                   | note
                   | silence

  note             : NOTA LPAREN pitch COMMA numeric_value COMMA duration RPAREN SEMICOLON

  silence          : SILENCIO LPAREN duration RPAREN SEMICOLON

  repetition       : REPETIR LPAREN numeric_value RPAREN LBRACKET voice_content RBRACKET

  numeric_value    : NUMBER
                   | ID

  duration         : NOTE_VALUE
                   | NOTE_VALUE POINT

  pitch            : MUSICAL_NOTE
                   | MUSICAL_NOTE PLUS
                   | MUSICAL_NOTE MINUS
```

3 Implementación.

3.1 Lexer

La regla del token ID incluye la generación de cualquier cadena alfanumerica. Muchas keyword del lexer, son también cadenas alfanuméricas, por ejemplo: `TEMPO : 'tempo'`, `COMPAS : 'compas'`, o cualquiera de las notas musicales (do re mi fa sol la si).
Para poder distinguir las palabras reservadas del lenguaje de cualquier otra cadena alfanumérica que se presente, se realizó la siguiente distinción en el lexer.

```
keywords = {
    'tempo'    : 'TEMPO',
    'compas'   : 'COMPAS',
    'const'    : 'CONST',
    'voz'      : 'VOZ',
    'repetir'  : 'REPETIR',
    'nota'     : 'NOTA',
    'silencio' : 'SILENCIO',
}
def t_ID(token):
    r'[_a-zA-Z][_a-zA-Z0-9]*'
    if token.value in keywords:   # Check for keywords
        token.type = keywords.get(token.value)
    elif token.value in musical_notes:
        token.type = 'MUSICAL_NOTE'
    elif token.value in note_values:
        token.type = 'NOTE_VALUE'
    else:
        token.type = 'ID'
    return token
```

Se observa que la regla ID chequea que la cadena alfanumérica reconocida sea o no, una palabra reservada.

3.2 Parser

Durante la fase de parsing se ejecuta el parser generado mediante la librería PLY. Cada producción de la gramática planteada para la resolución del ejercicio se modela mediante un árbol de parsing con exactamente la misma estructura reconocida por el parser. Por ejemplo:
Producción `note`, la definimos en PLY como:

```
def p_note(subs):
    'note : NOTA LPAREN pitch COMMA numeric_value COMMA duration RPAREN SEMICOLON'
    subs[0] = Note(subs[3], subs[5], subs[7], line=subs.lineno(1))
```

Donde,`Note` es una clase que expresa el comportamiento de los nodos del tipo `Note`. Por simplicidad y modularidad, existe aproximadamente, una subclase de `Node` por cada producción de la gramática. Por ejemplo, las producciones silence, duration se corresponden con las subclases de `Node` Silence y Duration. La libería PLY permite asociar acciones semánticas a cada producción.

3.3 Análisis Semántico

Mediante la clase `MusicLengSemanticAnalizer` se implementa el control semántico necesario para el lenguaje.

- Chequeo de declaración de constantes: No se puede redeclarar símbolos de constante
- Chequeo de referencias de constantes no declaradas
- Control de que los instrumentos válidos estén en el rango de 0 a 127
- Repeat(0) no aceptado.
- Chequeo de la correcta duración
- Octava debe ser un valor de 1 a 9

3.4 Encoder

Mediante la clase `SMFEncoder` se recorre el AST generado y se genera el archivo intermedio, en el formato necesario para poder convertirlo a MIDI. Se utilizó un volumen por default de 70.

4 Requerimientos

- Python 3.3 o superior
- Librería PLY 3.6 (`http://www.dabeaz.com/ply/`)
- Midicomp (`https://github.com/markc/midicomp/`)

5 Modo de uso

Para hacer uso del programa simplemente hay que ejecutar el archivo musileng en directorio raiz pasándole en el primer parámetro el archivo de entrada y el nombre del archivo que se desea generar en el segundo.
Ejemplo: `./musileng ejemplos/ej1_input.txt ejemplos/ej1_output.txt`

5.1 Tests

Para correr la test suite hay que ejecutar dentro del directorio ./src, donde se encuentra el código fuente, el comando `python -m unittest discover tests '*_test.py'`

6 Ejemplos de árboles de derivación

La entrada a procesar es la siguiente:

```
#tempo redonda 60
#compas 2/2
const grand_piano = 10;
voz (grand_piano)
{
  compas
  {
    nota(fa, 5, redonda);
  }
}
```

A continuación se muestra el árbol de derivación resultante. Por motivos de simplicidad en la visualización y para mayor comprensión, el árbol se muestra por partes. En la primera parte se observa la regla principal, musileng y su respectiva derivación. Luego las reglas correspondientes a `constants` y a `voices` se muestran en las sucesivas figuras

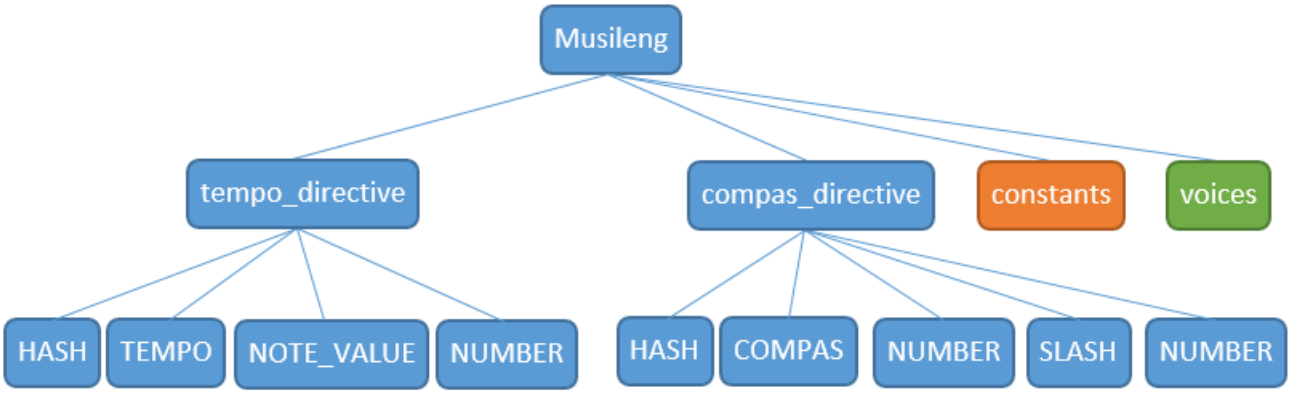

Figure 1: Árbol de derivación: Primera Parte - Regla: musileng

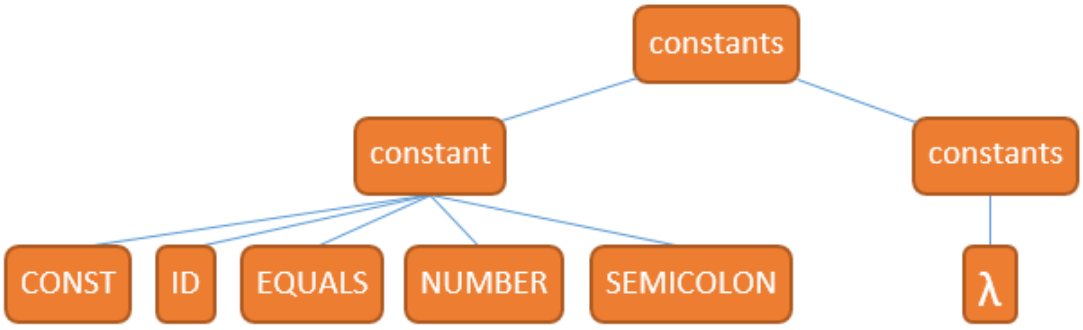

Figure 2: Árbol de derivación: Segunda Parte - Regla: constants

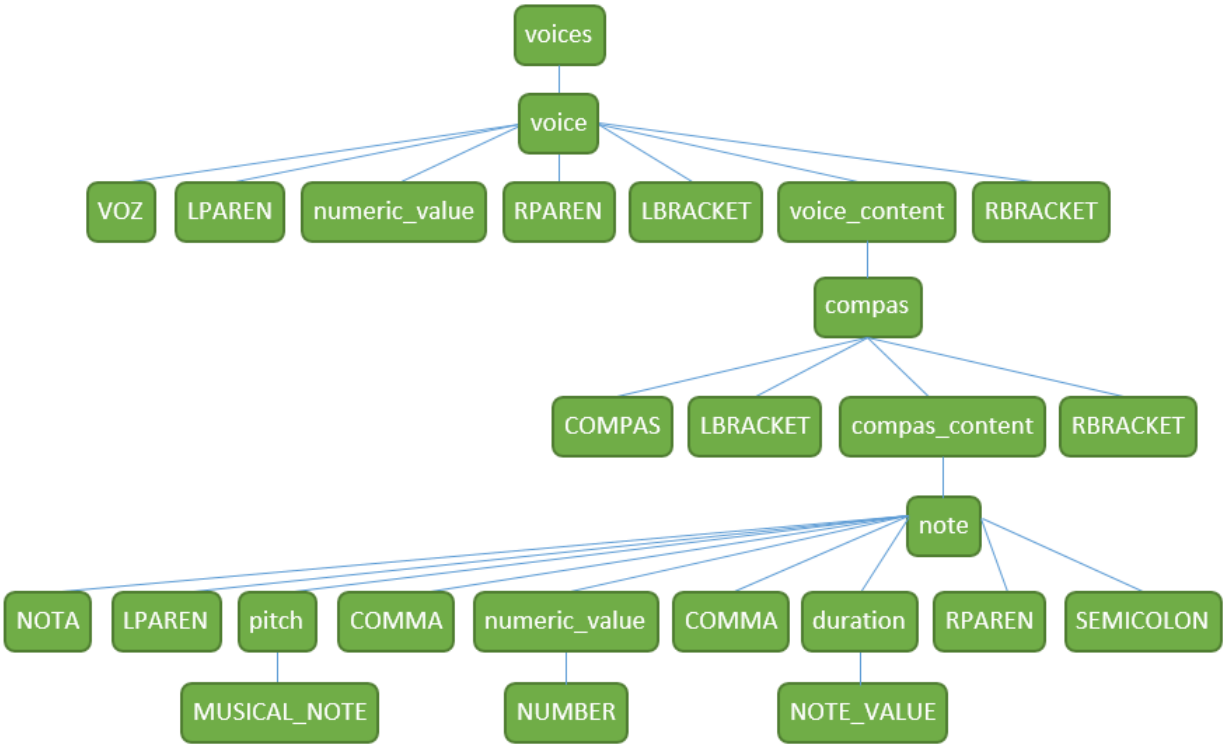

Figure 3: Árbol de derivación: Tercera Parte - Regla: voices

7 Ejemplos de ejecución

Presentamos a continuación algunos ejemplos que permiten ver la salida del programa

7.1 Entrada válida 1

Archivo de Entrada:

```
#tempo redonda 60
#compas 2/2
const octava = 5;
const grand_piano = 6;
voz (grand_piano)
{
  compas
  {
    nota(si, octava, blanca);
    silencio(blanca);
  }
}
```

Archivo de Salida:

```
MFile 1 2 384
MTrk
000:00:000 TimeSig 2/2 24 8
000:00:000 Tempo 250000
000:00:000 Meta TrkEnd
TrkEnd
MTrk
000:00:000 Meta TrkName "Voz 1"
000:00:000 ProgCh ch=1 prog=6
000:00:000 On  ch=1 note=b5  vol=70
000:01:000 Off ch=1 note=b5  vol=0
001:00:000 Meta TrkEnd
TrkEnd
```

7.2 Entrada válida 2

Archivo de Entrada:

```
// 60 redondas por minuto, osea cada redonda dura 1 seg.
#tempo redonda 60

#compas 2/2
const octava = 5;
const grand_piano = 0;
voz (grand_piano)
{
  compas
  {
    nota(do, octava, blanca);
    nota(re, octava, blanca);
  }
  compas
  {
    nota(mi, octava, blanca);
    nota(fa, octava, blanca);
  }
  compas
  {
    nota(sol-, octava, negra);
    nota(sol, octava, negra);
    nota(sol+, octava, negra);
    nota(la, octava, negra);
  }
  compas
  {
    nota(si, octava, blanca);
    // esto es un silencio que dura una blanca.
    silencio(blanca);
  }
}
```

Archivo de Salida:

```
MFile 1 2 384
MTrk
000:00:000 TimeSig 2/2 24 8
000:00:000 Tempo 250000
000:00:000 Meta TrkEnd
TrkEnd
MTrk
000:00:000 Meta TrkName "Voz 1"
000:00:000 ProgCh ch=1 prog=0
000:00:000 On  ch=1 note=c5  vol=70
000:01:000 Off ch=1 note=c5  vol=0
000:01:000 On  ch=1 note=d5  vol=70
001:00:000 Off ch=1 note=d5  vol=0
001:00:000 On  ch=1 note=e5  vol=70
001:01:000 Off ch=1 note=e5  vol=0
001:01:000 On  ch=1 note=f5  vol=70
002:00:000 Off ch=1 note=f5  vol=0
002:00:000 On  ch=1 note=g-5 vol=70
002:00:192 Off ch=1 note=g-5 vol=0
002:00:192 On  ch=1 note=g5  vol=70
002:01:000 Off ch=1 note=g5  vol=0
002:01:000 On  ch=1 note=g+5 vol=70
002:01:192 Off ch=1 note=g+5 vol=0
002:01:192 On  ch=1 note=a5  vol=70
003:00:000 Off ch=1 note=a5  vol=0
003:00:000 On  ch=1 note=b5  vol=70
003:01:000 Off ch=1 note=b5  vol=0
004:00:000 Meta TrkEnd
TrkEnd
```

7.3 Entrada inválida 1

Todas las constantes deben declararse antes de ser utilizadas.

```
#tempo redonda 60
#compas 2/2
const octava = 5;
voz (grand_piano) // Error de validacion: constante no definida (grand_piano)
{
  compas
  {
    nota(si, octava, blanca);
    silencio(blanca);
  }
}
```

Salida: `Ocurrió un error en la línea 6:  'grand_piano' no fue declarada`

7.4 Entrada inválida 2

Las constantes no pueden redefinirse en el programa.

```
#tempo redonda 60
#compas 2/2
const octava = 5;
const grand_piano = 8;  // Definimos la constante grand_piano
const grand_piano = 7;  // Error: Las constantes no pueden re-definirse
voz (grand_piano)
{
  compas
  {
    nota(si, octava, blanca);
    silencio(blanca);
  }
}
```

Salida: `Ocurrió un error en la línea 5:  'grand_piano' ya fue declarada`

7.5 Entrada inválida 3

Detección de errores de validación en los tiempos de un compás

```
// 60 redondas por minuto, osea cada redonda dura 1 seg.
#tempo redonda 60
// compas de 2 tiempos y cada tiempo dura 1/2 redonda (1 blanca).
// entonces cada compas puede tener 1 redonda, 2 blancas, 4 negras, etc.
#compas 2/2

const octava = 5;
const grand_piano = 8;

voz (grand_piano)
{
  compas // Error de validacion: el compas tiene tiempos de mas
  {
    nota(mi, octava, redonda);
    nota(fa, octava, redonda);
  }
  compas
  {
    nota(sol-, 11, negra); // Octava invalida: fuera de rango
    nota(sol, octava, negra);
    nota(sol+, octava, negra);
    nota(la, octava, negra);
  }
}
```

Salida:

```
Ocurrió un error en la línea 12:  compás de 3/2 declarado pero el configurado era de 1
```

7.6 Entrada inválida 4

Detección de octavas no válidas

```
#tempo redonda 60
#compas 2/2

const octava = 5;
const grand_piano = 8;

voz (grand_piano)
{
  compas
  {
    nota(fa, octava, redonda);
  }
  compas
  {
    nota(sol-, 11, negra); // Octava invalida: fuera de rango
    nota(sol, octava, negra);
    nota(sol+, octava, negra);
    nota(la, octava, negra);
  }
}
```

Salida: `Ocurrió un error en la línea 15:  '11' no es una octava válida`

7.7 Entrada inválida 5

La estructura `repeat(veces)` acepta números o constantes para facilitar el trabajo del programador de musileng. Pero es un error querer iterar 0 veces con un repeat.

```
#tempo redonda 60
#compas 2/2

const octava = 5;
const grand_piano = 8;
const a = 0;

voz (grand_piano)
{
  repetir(a) // Error: No es válido repetir 0 veces
  {
    compas
    {
      nota(fa, octava, redonda);
    }
    compas
    {
      nota(sol-, 9, negra);
      nota(sol, octava, negra);
      nota(sol+, octava, negra);
      nota(la, octava, negra);
    }
  }
}
```

Salida: `Ocurrió un error en la línea 10:  no es válida una repetición de 0 iteraciones`

8 Decisiones de Diseño

A continuación se enumeran las distintas decisiones de diseño que se tomaron en el transcurso del desarrollo del parser.

- Sobre el control de errores en el archivo de entrada se reporta la falla del primer error encontrado
- Interpretamos que redefinir una constante es una falla del archivo de entrada, por lo tanto realizar este tipo de acción genera un mensaje de error.

- La estructura `repetir(veces)` puede aceptar tanto un número literal, como una constante. El caso particular de `repetir(0)`, fue considerado un error en el archivo de entrada, por lo tanto, termina su ejecución y muestra un error al detectar este caso.
- Los errores semánticos incluyen número de línea para facilitar la corrección por parte del programador/compositor de musileng

9 Código Fuente

9.1 parser.py

```
from musileng.lexer import tokens
from musileng.ast import *


def p_musileng(subs):
    'musileng : tempo_directive compas_directive constants voices'
    subs[0] = MusiLeng(*subs[1:5])


def p_tempo_directive(subs):
    'tempo_directive : HASH TEMPO NOTE_VALUE NUMBER'
    subs[0] = TempoDirective(subs[3], subs[4], line=subs.lineno(1))

def p_compas_directive(subs):
    'compas_directive : HASH COMPAS NUMBER SLASH NUMBER'
    subs[0] = BarDirective(subs[3], subs[5], line=subs.lineno(1))


def p_constants(subs):
    """constants : constant constants
                 | """
    subs[0] = [subs[1]] + subs[2] if len(subs) == 3 else []

def p_constant(subs):
    'constant : CONST ID EQUALS NUMBER SEMICOLON'
    subs[0] = ConstDecl(subs[2], subs[4], line=subs.lineno(1))


def p_voices(subs):
    '''voices : voice voices
              | voice'''
    subs[0] = [subs[1]] + (subs[2] if len(subs) == 3 else [])


def p_voice(subs):
    'voice : VOZ LPAREN numeric_value RPAREN LBRACKET voice_content RBRACKET'
    subs[0] = Voice(subs[3], subs[6], line=subs.lineno(1))

def p_voice_content(subs):
    '''voice_content : compas voice_content
                     | repetition voice_content
                     | compas
                     | repetition'''
    subs[0] = [subs[1]] + (subs[2] if len(subs) == 3 else [])


def p_compas(subs):
    'compas : COMPAS LBRACKET compas_content RBRACKET'
    subs[0] = Bar(subs[3], line=subs.lineno(1))


def p_compas_content(subs):
    '''compas_content : note compas_content
                      | silence compas_content
                      | note
                      | silence'''
    subs[0] = [subs[1]] + (subs[2] if len(subs) == 3 else [])

def p_note(subs):
    'note : NOTA LPAREN pitch COMMA numeric_value COMMA duration RPAREN SEMICOLON'
    subs[0] = Note(subs[3], subs[5], subs[7], line=subs.lineno(1))

def p_silence(subs):
```

```
    'silence : SILENCIO LPAREN duration RPAREN SEMICOLON'
    subs[0] = Silence(subs[3], line=subs.lineno(1))


def p_repetition(subs):
    'repetition : REPETIR LPAREN numeric_value RPAREN LBRACKET voice_content RBRACKET'
    subs[0] = Repeat(subs[3], subs[6], line=subs.lineno(1))


def p_numeric_value(subs):
    '''numeric_value : NUMBER
                     | ID'''
    if type(subs[1]) == int:
        subs[0] = Literal(subs[1], line=subs.lineno(1))
    else:
        subs[0] = ConstRef(subs[1], line=subs.lineno(1))

def p_duration(subs):
    '''duration : NOTE_VALUE
                | NOTE_VALUE POINT'''
    subs[0] = Duration(subs[1], len(subs) == 3, line=subs.lineno(1))


def p_pitch(subs):
    '''pitch : MUSICAL_NOTE
             | MUSICAL_NOTE PLUS
             | MUSICAL_NOTE MINUS'''
    if len(subs) == 3:
        subs[0] = Pitch(subs[1], subs[2], line=subs.lineno(1))
    else:
        subs[0] = Pitch(subs[1], line=subs.lineno(1))


def p_error(token):
    message = "[Error de Sintaxis]"
    if token is not None:
        message += " token " + str(token.value)
        message += " inesperado en línea " + str(token.lineno)
    else:
        message += " fin de archivo inesperado"
    raise SyntaxError(message)
```

9.2 lexer.py

```
import re

keywords = {
    'tempo'    : 'TEMPO',
    'compas'   : 'COMPAS',
    'const'    : 'CONST',
    'voz'      : 'VOZ',
    'repetir'  : 'REPETIR',
    'nota'     : 'NOTA',
    'silencio' : 'SILENCIO',
}

musical_notes = ['do', 're', 'mi', 'fa', 'sol', 'la', 'si']
note_values = ['redonda', 'blanca', 'negra', 'corchea', 'semicorchea', 'fusa', 'semifusa']


tokens = (
    # Operators
    'HASH',
    'EQUALS',
    'SLASH',
    'POINT',
    'PLUS',
    'MINUS',

    # Delimiters
    'SEMICOLON',
    'COMMA',
    'LPAREN',
    'RPAREN',
    'LBRACKET',
    'RBRACKET',

    # Misc
    'MUSICAL_NOTE',
    'NOTE_VALUE',
    'NUMBER',
    'ID',
) + tuple(keywords.values())


def t_NUMBER(token):
    r'[0-9]+'
    token.value = int(token.value)
    return token

def t_ID(token):
    r'[_a-zA-Z][_a-zA-Z0-9]*'
    if token.value in keywords:   # Check for keywords
        token.type = keywords.get(token.value)
    elif token.value in musical_notes:
        token.type = 'MUSICAL_NOTE'
    elif token.value in note_values:
        token.type = 'NOTE_VALUE'
    else:
        token.type = 'ID'

    return token


t_HASH   = r'\#'
t_EQUALS = r'='
t_SLASH  = r'/'
t_POINT  = r'\.'
t_PLUS   = r'\+'
t_MINUS  = r'\-'
```

```
t_SEMICOLON = r';'
t_COMMA  = r','
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACKET = r'\{'
t_RBRACKET = r'\}'


t_ignore = " \t"

def t_COMMENT(token):
    r'//[^\n]*\n'
    token.lexer.lineno += 1

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


def t_error(token):
    message = "[Error de Sintaxis] Token desconocido "
    message += re.sub(r'\s+.*', '', str(token.value))
    message += " en línea " + str(token.lineno)
    raise SyntaxError(message)
```

9.3 ast.py

```
from fractions import Fraction

class Node:
    def __init__(self, *args, line=None, **kargs):
        self.lineno = line
        self.init_subnodes(*args, **kargs)

    def accept(self, visitor):
        getattr(visitor, 'visit_type_' + type(self).__name__)(self)

    def init_subnodes(self, *args, **kargs): pass

class MusiLengError(Exception):
    def __init__(self, node, detail):
        msg = "Ocurrió un error"
        if node.lineno:
            msg += " en la línea {}".format(node.lineno)
        super().__init__(msg + ": "+ detail)


class MusiLeng(Node):
    def init_subnodes(self, tempo_dir, bar_dir, consts, voices):
        if len(voices) > 16: raise TooManyVoices(voices[16])
        self.tempo, self.bar, self.consts, self.voices = tempo_dir, bar_dir, consts, voices

    def voices_number(self):
        return len(self.voices)

class TooManyVoices(MusiLengError):
    def __init__(self, node):
        super().__init__(node, 'no se pueden definir más de 16 voces')


class Literal(Node):
    def init_subnodes(self, number):
        self.number = number

    def value(self, symbol_table = {}):
        return self.number

class ConstRef(Node):
    def init_subnodes(self, identifier):
        self.identifier = identifier

    def value(self, symbol_table):
        return symbol_table.get(self.identifier)


class TempoDirective(Node):
    def init_subnodes(self, note_value, notes_per_min):
        if notes_per_min == 0: raise InvalidTempo(self)
        self.reference_note, self.notes_per_min = Duration(note_value), notes_per_min

    def microseconds_per_quarter_note(self):
        return int((1000000 * 60 * self.reference_note.note_value_number()) /
                   (4 * self.notes_per_min))

class BarDirective(Node):
    valid_note_values = [ 2**i for i in range(0,7) ]

    def init_subnodes(self, pulses, note_value_number):
        if pulses == 0: raise InvalidBarPulses(self)
        if not note_value_number in self.valid_note_values: raise InvalidBarBase(self)
        self.pulses, self.note_value_number = pulses, note_value_number

    def pulse_duration(self):
```

```
        return Duration.from_number(self.note_value_number)

    def fraction(self):
        return Fraction(self.pulses, self.note_value_number)

    def formated(self):
        return "%d/%d" % (self.pulses, self.note_value_number)

class InvalidTempo(MusiLengError):
    def __init__(self, node):
        super().__init__(node, 'la cantidad de notas por minuto en tempo no puede ser 0')

class InvalidBarPulses(MusiLengError):
    def __init__(self, node):
        super().__init__(node, 'la cantidad de pulsos por compás no puede ser 0')

class InvalidBarBase(MusiLengError):
    def __init__(self, node):
        super().__init__(node, 'número de figura inválido para el compás')


class ConstDecl(Node):
    def init_subnodes(self, identifier, number):
        self.identifier, self.number = identifier, number


class Note(Node):
    def init_subnodes(self, pitch, octave, duration):
        self.pitch, self.octave, self.duration = pitch, octave, duration

class Silence(Node):
    def init_subnodes(self, duration):
        self.duration = duration


class Duration(Node):
    numbers = {
        'redonda':      1,
        'blanca':       2,
        'negra':        4,
        'corchea':      8,
        'semicorchea':  16,
        'fusa' :        32,
        'semifusa':     64,
    }
    proportions = { name : Fraction(1, num) for name, num in numbers.items() }

    @classmethod
    def from_number(cls, number):
        for name, num in cls.numbers.items():
            if num == number: return Duration(name)

    def init_subnodes(self, note_value, dotted=False):
        self.note_value, self.dotted = note_value, dotted

    def note_value_number(self):
        return self.numbers[self.note_value]

    def fraction(self):
        return self.proportions[self.note_value] * self.dot_increase()

    def dot_increase(self):
        return Fraction(3,2) if self.dotted else 1

class Pitch(Node):
    american_codes = {'do' : 'c', 're' : 'd', 'mi' : 'e', 'fa' : 'f', 'sol' : 'g', 'la' : 'a', 'si' : 'b'}
```

```
    def init_subnodes(self, musical_note, modifier=None):
        self.musical_note, self.modifier = musical_note, modifier

    def american(self):
        return self.american_codes[self.musical_note] + (self.modifier if self.modifier else '')

class Voice(Node):
    def init_subnodes(self, instrument, childs):
        self.instrument, self.childs = instrument, childs

class Repeat(Node):
    def init_subnodes(self, times, childs):
        self.times, self.childs = times, childs

class Bar(Node):
    def init_subnodes(self, notes):
        self.notes = notes

    def duration(self):
        return sum(note.duration.fraction() for note in self.notes)
```

9.4 encoder.py

```
from musileng.ast import *
from musileng.visitor import *
from musileng.semantic_analysis import SemanticVisitor

class SMFEncoder(SemanticVisitor):
    default_volume = 70

    def __init__(self, output):
        self.output = output
        super().__init__()

    def visit_type_MusiLeng(self, node):
        self.visit(node.tempo)
        self.visit(node.bar)
        self.visit_all(node.consts)
        self.output_header_track(node.voices_number())

        for number, voice in enumerate(node.voices):
            self.current_track = number + 1
            self.visit(voice)

    def visit_type_TempoDirective(self, node):
        self.quarter_note_length = node.microseconds_per_quarter_note()

    def visit_type_BarDirective(self, node):
        self.setup_midi_timer(node.pulses, node.pulse_duration())
        self.formated_bar = node.formated()

    def visit_type_Voice(self, node):
        self.timer.reset()

        self.output_track_begin()
        self.output_timestamped_line('Meta TrkName "Voz {NUMERO_DE_VOZ}"', NUMERO_DE_VOZ=self.current_track)
        self.output_timestamped_line('ProgCh ch={CANAL} prog={INSTRUMENTO}', CANAL=self.current_track, INSTRUMENTO=s

        self.visit_all(node.childs)

        self.output_timestamped_line('Meta TrkEnd')
        self.output_track_end()

    def visit_type_Repeat(self, node):
        for n in range(self.value(node.times)):
            self.visit_all(node.childs)

    def visit_type_Note(self, node):
        self.output_timestamped_line('On  ch={CANAL} note={NOTA} vol={VOL}', CANAL=self.current_track, NOTA=self.for
        self.timer.elapse_by(node.duration)
        self.output_timestamped_line('Off ch={CANAL} note={NOTA} vol={VOL}', CANAL=self.current_track, NOTA=self.for

    def visit_type_Silence(self, node):
        self.timer.elapse_by(node.duration)


    def setup_midi_timer(self, pulses, pulse_duration):
        self.timer = MidiTimer(pulses, pulse_duration)

    def output_header_track(self, voices):
        self.output_line("MFile 1 {NTRACKS} 384", NTRACKS=voices + 1)
        self.output_track_begin()
        self.output_timestamped_line("TimeSig {COMPAS} 24 8", COMPAS=self.formated_bar)
        self.output_timestamped_line("Tempo {TEMPO}", TEMPO=self.quarter_note_length)
        self.output_timestamped_line("Meta TrkEnd")
        self.output_track_end()

    def output_track_begin(self):
        self.output_line("MTrk")
```

```
    def output_track_end(self):
        self.output_line("TrkEnd")

    def output_timestamped_line(self, string, **kargs):
        self.output_line(self.timer.formated() + ' ' + string, **kargs)

    def output_line(self, string, **kargs):
        self.output.write(string.format(**kargs))
        self.output.write("\n")

    def format_note(self, note):
        return (note.pitch.american() + str(self.value(note.octave))).ljust(3)

class MidiTimer:
    clicks_per_pulse = 384

    def __init__(self, pulses, pulse_duration):
        self.clicks_per_full_note = self.clicks_per_pulse * pulse_duration.note_value_number()
        self.bar_pulses = pulses
        self.reset()

    def reset(self):
        self.bar = self.pulse = self.clicks = 0

    def elapse_by(self, duration):
        pulses, self.clicks = divmod(self.clicks + self.clicks_per_full_note * duration.fraction(), self.clicks_per_
        bars, self.pulse = divmod(self.pulse + pulses, self.bar_pulses)
        self.bar += bars

    def formated(self):
        return "%03d:%02d:%03d" % (self.bar, self.pulse, self.clicks)

class Track10MustBeAPercussionInstrument(Exception):
    def __init__(self, instrument):
        super().__init__("El instrumento número {} asignado al track 10 no es de percusión según la especificación M
```

9.5 visitor.py

```
from musileng.ast import *

class Visitor:
    def visit(self, node):
        node.accept(self)

    def visit_all(self, nodes):
        for node in nodes:
            self.visit(node)

    def visit_type_MusiLeng(self, node):
        self.visit(node.tempo)
        self.visit(node.bar)
        self.visit_all(node.consts)
        self.visit_all(node.voices)

    def visit_type_TempoDirective(self, node): pass

    def visit_type_BarDirective(self, node): pass

    def visit_type_ConstDecl(self, node): pass

    def visit_type_Voice(self, node):
        self.visit(node.instrument)
        self.visit_all(node.childs)

    def visit_type_Repeat(self, node):
        self.visit(node.times)
        self.visit_all(node.childs)

    def visit_type_Bar(self, node):
        self.visit_all(node.notes)

    def visit_type_Note(self, node):
        self.visit(node.pitch)
        self.visit(node.octave)
        self.visit(node.duration)

    def visit_type_Silence(self, node):
        self.visit(node.duration)

    def visit_type_Duration(self, node): pass

    def visit_type_Pitch(self, node): pass

    def visit_type_Literal(self, node): pass

    def visit_type_ConstRef(self, node): pass
```

9.6 semantic_analysis.py

```
from musileng.ast import *
from musileng.visitor import *

class SemanticVisitor(Visitor):
    def __init__(self):
        self.symbol_table = {}

    def visit_type_ConstDecl(self, node):
        self.symbol_table[node.identifier] = node.number

    def value(self, numeric_node):
        return numeric_node.value(self.symbol_table)


class MusiLengSemanticAnalizer(SemanticVisitor):
    def visit_type_BarDirective(self, node):
        self.bar_duration = node.fraction()

    def visit_type_ConstDecl(self, node):
        if node.identifier in self.symbol_table:
            raise RedeclaredSymbol(node)
        else:
            super().visit_type_ConstDecl(node)

    def visit_type_ConstRef(self, node):
        if not node.identifier in self.symbol_table:
            raise UndeclaredSymbol(node)

    def visit_type_Voice(self, node):
        super().visit_type_Voice(node)
        instrument = self.value(node.instrument)
        if not 0 <= instrument <= 127: raise InvalidInstrument(node.instrument, instrument)

    def visit_type_Repeat(self, node):
        super().visit_type_Repeat(node)
        if self.value(node.times) == 0: raise InvalidRepeatNumber(node.times)

    def visit_type_Bar(self, node):
        super().visit_type_Bar(node)
        if node.duration() != self.bar_duration:
            raise InvalidBarDuration(node, self.bar_duration, node.duration())

    def visit_type_Note(self, node):
        super().visit_type_Note(node)
        octave = self.value(node.octave)
        if not 1 <= octave <= 9: raise InvalidOctave(node.octave, octave)


class RedeclaredSymbol(MusiLengError):
    def __init__(self, node):
        super().__init__(node, "'{}' ya fue declarado previamente".format(node.identifier))

class UndeclaredSymbol(MusiLengError):
    def __init__(self, node):
        super().__init__(node, "'{}' no fue declarado".format(node.identifier))

class InvalidRepeatNumber(MusiLengError):
    def __init__(self, node):
        super().__init__(node, "no es válida una repetición de 0 iteraciones")


class InvalidOctave(MusiLengError):
    def __init__(self, node, number):
        super().__init__(node, "'{}' no es una octava válida".format(number))

class InvalidInstrument(MusiLengError):
```

```
    def __init__(self, node, number):
        super().__init__(node, "'{}' no es un instrumento válido".format(number))

class InvalidBarDuration(MusiLengError):
    def __init__(self, node, midi_bar, invalid_bar):
        super().__init__(node, "compás de '{invalid}' declarado pero el configurado era de {bar}".format(bar=midi_ba
```

9.7 cli.py

```
from argparse import ArgumentParser
from ply import lex, yacc

import musileng.parser
import musileng.lexer
from musileng.ast import MusiLengError
from musileng.semantic_analysis import MusiLengSemanticAnalizer
from musileng.encoder import SMFEncoder


if __name__ == "__main__":
    cli_parser = ArgumentParser(description='Generador de archivos SMF')
    cli_parser.add_argument('entrada', help='nombre del archivo fuente')
    cli_parser.add_argument('salida', help='nombre del archivo generado')
    args = cli_parser.parse_args()

    try:
        with open(args.entrada, 'r') as src:
            lexer = lex.lex(module=musileng.lexer)
            parser = yacc.yacc(module=musileng.parser)
            analizer = MusiLengSemanticAnalizer()

            mus = parser.parse(src.read(), lexer)
            analizer.visit(mus)

            with open(args.salida, "w") as dest:
                encoder = SMFEncoder(dest)
                encoder.visit(mus)
    except FileNotFoundError as ioe:
        exit('No se pudo acceder al archivo: ' + ioe.filename)
    except MusiLengError as me:
        exit(str(me))
```