DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Miércoles 29 de julio de 2015 — Teoría de Lenguajes

Integrante	LU	Correo electrónico
Aleman, Damián Eliel	377/10	damian_8591@hotmail.com
Gauna, Claudio Andrés	733/06	gauna_claudio@yahoo.com.ar

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

Índice

1. Introducción

El objetivo del trabajo practico es implementar un parser para un lenguaje orientado a la composicion de piezas musicales, llamado Musileng, que luego sera transformado al formato MIDI 1 para su reproduccion.
Los pasos que seguimos para realizar el trabajo practico fueron:

- Generar la gramatica adecuada, que sintetice el lenguaje orientado a la composicion de las piezas musicales
- Escribir los terminales del lenguaje y las reglas del lexer
- Escribir los no termianles del lengueje y las reglas del parser.
- Agregarle semantica para que pueda imprimir al lenguaje intermedio que pueda ser leido por el programa midcomp
- Luego de finalizada la traduccion, pueda transformarse a MIDI (.mid) por medio del programa midicomp

Utilizamos para el trabajo la herramienta ANTLR para generar el parser (y el lexer) necesitado.

2. Grámatica

Primero mostramos la gramatica del lexer que lee el archivo como secuencia de caracteres y los mapea a una secuencia de simbolos terminales.
El conjunto de terminales es: {`NUMERAL, TEMPO, DURACION, NUM, COMPAS, SLASH, CONST, NOMBRE, IGUAL, VOZ, LPAREN, RPAREN, LBRACE, RBRACE, REPETIR, SILENCIO, PUNTILLO, ALTURA, OCTAVA, COMA, PUNTOYCOMA` }

A continuación definimos la gramática que generamos para sintetizar el lenguaje: Las terminales de la gramática son todas las cadenas que están entre comillas simples y los que se derivan a partir de una regla (de los tokens en mayúscula).

2.1. Lexer

A la gramatica del lexer, le dimos reglas para que pueda reconocer los tokens:
TEMPO→ 'tempo';
COMPAS → 'compas';
CONST→ 'const';
IGUAL→ '=';
VOZ→ 'voz';
LPAREN→ '(' ;
RPAREN→ ')' ;
LBRACE→ '' ;
RBRACE→ '' ;
NUMERAL→ '#';
COMA→ ',' ;
PUNTOYCOMA→ ';';
SLASH→ '/';
REPETIR→ 'repetir';
SILENCIO→ 'silencio';
NOTA→ 'nota';
PUNTILLO → '.';
DURACION → ('redonda' |'blanca'|'negra'|'corchea'|'semicorchea'|'fusa'|'semifusa');
ALTURA → ('do'| 're'| 'mi'| 'fa'| 'sol'| 'la'| 'si'|'do+'| 're+'| 'mi+'| 'fa+'| 'sol+'| 'la+'| 'si+'|
'do-'| 're-'| 'mi-'| 'fa-'| 'sol-'| 'la-'| 'si-');
NUM→ [0-9]+;
NOMBRE→ [a-zA-Z_]+;
OCTAVA→ [1-9];
COMENTARIOS→ '\\' ∼[\r\ n]* → skip ;
WS → [\t\r\n] + → skip ;

2.2. Parser

A continuación mostramos la gramatica del parser:
S → tempos elcompas constantes melodia
tempos → NUMERAL TEMPO DURACION NUM
elcompas → NUMERAL COMPAS NUM SLASH NUM
constantes → constante*
constante → CONST NOMBRE IGUAL NUM
melodia → (VOZ LPAREN LPAREN NUM RPAREN { compases})+
| (VOZ LPAREN texto RPAREN { compases})+
compases → compas compases | repeticion compases
repeticion → REPETIR LPAREN NUM RPAREN LBRACE compas RBRACE
compas → (COMPAS LBRACE nota RBRACE)+ | (COMPAS LPAREN silencio RPAREN)+
silencio → SILENCIO LPAREN DURACION PUNTILLO? RPAREN
nota → NOTA LPAREN ALTURA COMA octava, DURACION PUNTILLO? RPAREN
octava → OCTAVA | NOMBRE

Notar que los símbolos en mayúsculas son terminales, mientras los que estan en minúscula son no terminales.
Por decision de diseño, el lenguaje no admite una repeticion anidada dentro de otra ya que creemos que no es necesario.
El símbolo distinguido es S.

El conjunto de no terminales es:
{`S, tempos, elcompas,contantes, contante, melodia, compases, repeticion, compas, silencio,nota, octava`}

2.3. Atributos

Asignamos atributos para verificar las restricciones que tenemos que hacer de modo tal que la gramática genere el lenguaje que necesitamos. Los atributos también los usaremos para realizar la traducción al lenguaje intermedio para que sea legible por el programa midcomp.

Los atributos sintetizados son: {`partitura, tempo, indicacion, listaCompases, voces, repeticiones, compasObj, silencioObj,notaObj, valor`}

Los atributos heredados son: {`indicacion`}
Ahora haremos una breve explicación de cada atributo:

- *partitura*: Es el objeto raiz, almacena el tempo, la indicación y la lista de voces.
- *tempo*: Este atributo guarda la información de la duración de la figura y la cantidad de veces que entra esa figura en un minuto.
- *indicacion*: Almacena el numerador y el denominador definidos del compás.
- *voces*:La lista de voces de la melodía.
- *listaCompases*: La lista de los compasObj y las repetcionesObj.
- *compasObj*: Tiene la lista de notas de cada compas.
- *repeticiones*: Almacena la cantidad de repeticiones.
- *notaObj*: Almacena la altura,de que octava es, la duración, la alteración y si tiene puntillo.
- *silencioObj*: Tiene la lista de silencios de cada compas.
- *valor*: El valor de la octava .
- *listaCompases*: Tiene todos los compases definidos.

3. Midicomp

Finalmente, realizamos la traducción en un archivo llamado *midicomp.java*, que calcula los tiempos a partir del tempo y vamos prendiendo y apagando los canales por cada nota o silencio que hay en la melodía. Luego además del encabezado inicial que indica la cantidad de tracks y cantidad de ticks por negra, realizamos la traducción para todos los compases de cada voz que hay en el archivo de entrada.

4. Tests

4.1. Test Inválidos

Realizamos una serie de tests para verificar que el parser funciona correctamente. Para ello le dimos como entrada al parser, archivos invalidos:

- octava 10, es decir fuera del rango del 0 al 9.
- Repetir 0 veces una lista de compases.
- voces:La lista de voces de la melodia
- Distinta duracion de dos compases.
- Instrumento fuera de rango del 1 al 127.
- dos declaraciones distintas de una misma constante.
- Sin voz.

4.2. Test Válidos

Sobre los test válidos probamos el archivo de ejemplo del enunciado.
Otro test vimos que funcionaba correctamente el comando repetir y por último uno que utilizaba dos voces.

5. Conclusión

Con este trabajo pudimos llevar a la practica y automatizar un parser y un lexer para un lenguaje dado, utilizando la herramienta *ANTLR*. Así pudimos entender un poco mejor la diferencia entre los tokens que tiene que identificar el lexer y las sentencias
de no terminales que se pueden formar con esos tokens a partir de la gramatica. A continuación dejamos el código.

6. Apéndice

6.1. Código Lexer

```
lexer grammar lexerGrammar;

TEMPO: 'tempo';
COMPAS : 'compas';
CONST: 'const';
IGUAL: '=';
VOZ: 'voz';
LPAREN: '(' ;
RPAREN: ')' ;
LBRACE: '{' ;
RBRACE: '}' ;
NUMERAL:  '#';
COMA: ',' ;
PUNTOYCOMA: ';';
SLASH: '/';
REPETIR: 'repetir';
SILENCIO: 'silencio';
NOTA:  'nota';
PUNTILLO : '.';
DURACION : ('redonda'|'blanca'|'negra'|'corchea'|'semicorchea'|'fusa'|'semifusa');
ALTURA : ('do'| 're'| 'mi'| 'fa'| 'sol'| 'la'| 'si'|
  'do+'| 're+'| 'mi+'| 'fa+'| 'sol+'| 'la+'| 'si+'|
```

```
  'do-'| 're-'| 'mi-'| 'fa-'| 'sol-'| 'la-'| 'si-');

NUM: [0-9]+;
NOMBRE: [a-zA-Z_]+;
OCTAVA: [1-9] ;
COMENTARIOS:   '//' ~[\r\n]*  -> skip ;
//para escaper espacios, tabs, y saltos de linea
WS : [ \t\r\n]+ -> skip ;
```

6.2. Código Parser

```
grammar Musileng;

import lexerGrammar;

@parser::header {
import java.util.HashMap;
import java.util.Map;
import java.util.List;
import java.util.ArrayList;
}

@parser::members {
private Map<String,Integer> constantes = new HashMap<String,Integer>();

public static final int CLICKS_POR_PULSO = 384;

private boolean agregarConstante(String nombre, String valor) {
if (constantes.containsKey(valor)) {
constantes.put(nombre,constantes.get(valor));
return true;
} else {
return false;
}
}

public static int clicksPorFigura(String altura, IndicacionCompas indicacion) {
NotaEnum figura = NotaEnum.valueOf(altura);
int clicksPorRedonda = CLICKS_POR_PULSO * indicacion.tipoNota;
return clicksPorRedonda / Double.valueOf(4 / figura.getDuracion()).intValue();
}

private boolean agregarConstante(String nombre, Integer valor) {
if (constantes.containsKey(nombre)) {
return false;
}
constantes.put(nombre, valor);
return true;
}

private boolean validarRepeticiones(int valor) {
return valor > 0;
}

private boolean validarDuracion(List<Nota> notas, IndicacionCompas indicacion) {
Double duracion = 0.0;
for(Nota nota : notas) {
duracion += nota.calcularDuracion();
}
return duracion.compareTo(indicacion.duracion()) == 0;
}

private boolean validarAlMenosUnCompas(List<Compas> listaCompases) {
return !listaCompases.isEmpty();
```

```
}

private boolean validarInstrumento(int instrumento) {
return instrumento >= 0 && instrumento <= 127;
}

private boolean validarTempo(Tempo tempo) {
return true;
}

private void agregarRepetidos(List<Compas> listaCompases, List<Compas> nuevos, int repeticiones) {
for (int i = 0; i < repeticiones; i++) {
listaCompases.addAll(nuevos);
}
}

public static class IndicacionCompas {
public int tiempos;
public int tipoNota;
public IndicacionCompas(int tiempos, int tipoNota) {
this.tiempos = tiempos;
this.tipoNota = tipoNota;
}

public Double duracion() {
return tiempos * Double.valueOf(4 / Double.valueOf(tipoNota));
}
}

public static class Nota {
public String duracion;
public String altura;
public String alteracion;
public Integer octava;
boolean tienePuntillo;
public Nota(String altura, Integer octava, String duracion, boolean tienePuntillo) {
this.altura = altura;
this.octava = octava;
this.duracion = duracion;
this.alteracion = alteracion;
this.tienePuntillo = tienePuntillo;
}

public Double calcularDuracion() {
Double duracionNota = NotaEnum.valueOf(this.duracion).getDuracion();
return duracionNota + (this.tienePuntillo? duracionNota /2 : 0.0);
}

public boolean isSilencio() {
return altura == null;
}

public String notacionAmericana() {
return NotacionAmericana.notacionAmericana(this.altura)+octava;
}

public String toString() {
if (altura != null) {
return String.format("Nota(%s,%s,%s,%s,%s)", altura, octava, duracion, alteracion, tienePuntillo);
} else {
return String.format("Silencio(%s, %s)", this.duracion, this.tienePuntillo);
}
}
```

```
}

public enum NotaEnum {
redonda(1.0),blanca(2.0), negra(1.0),corchea(1/2.0),semicorchea(1/4.0),fusa(1/8.0),semifusa(1/16.0);

private Double duracion;

private NotaEnum(Double duracion) {
this.duracion = duracion;
}

public Double getDuracion() {
return this.duracion;
}
}

public static class NotacionAmericana {
private static Map<String,String> notacionAmericana = new HashMap<String,String>();
static {
notacionAmericana.put("do" , "c");
notacionAmericana.put("re" , "d");
notacionAmericana.put("mi" , "e");
notacionAmericana.put("fa" , "f");
notacionAmericana.put("sol" , "g");
notacionAmericana.put("la" , "a");
notacionAmericana.put("si" , "b");
notacionAmericana.put("do+" , "c+");
notacionAmericana.put("re+" , "d+");
notacionAmericana.put("fa+" , "f+");
notacionAmericana.put("sol+" , "g+");
notacionAmericana.put("la+" , "a+");
notacionAmericana.put("si+" , "b+");
notacionAmericana.put("do-" , "c-");
notacionAmericana.put("re-" , "d-");
notacionAmericana.put("mi-" , "e-");
notacionAmericana.put("fa-" , "f-");
notacionAmericana.put("sol-" , "g-");
notacionAmericana.put("la-" , "a-");
notacionAmericana.put("si-" , "b-");
}

public static String notacionAmericana(String altura) {
return notacionAmericana.get(altura);
}
}

public static class Compas {
public List<Nota> notas;
public Compas() {
this.notas = new ArrayList<Nota>();
}
public String toString() {
return notas.toString();
}
}

public static class Voz {
public int instrumento;
public List<Compas> compases;
public Voz(int instrumento, List<Compas> compases) {
this.instrumento = instrumento;
this.compases = compases;
}
public String toString() {
```

```
return String.format("Voz(instrumento = %s, compaces = {%s})", this.instrumento, this.compases);
}
}

public static class Melodia {
public List<Voz> voces;
public Melodia(List<Voz> voces) {
this.voces = voces;
}
}

public static class Tempo {
public String duracion;
public int cantidad;
public Tempo(String duracion, int cantidad) {
this.duracion = duracion;
this.cantidad = cantidad;
}

public String toString() {
return String.format("Tempo = (%s, %s)", cantidad, duracion);
}
}

public static class Partitura {
public Tempo tempo;
public IndicacionCompas indicacion;
public List<Voz> voces;
public Partitura(Tempo tempo, IndicacionCompas indicacion, List<Voz> voces) {
this.tempo = tempo;
this.voces = voces;
this.indicacion = indicacion;
}

public String toString() {
return String.format("%s; voces = %s", this.tempo, voces);
}
}
}

//Gramatica
s returns[Partitura partitura]: tempos elcompas constantes melodia[$elcompas.indicacion]
{$partitura = new Partitura($tempos.tempo, $elcompas.indicacion, $melodia.voces);};

tempos returns[Tempo tempo]: NUMERAL TEMPO DURACION NUM
{$tempo = new Tempo($DURACION.text, $NUM.int);}{validarTempo($tempo)}? ;

elcompas returns [IndicacionCompas indicacion] : NUMERAL COMPAS n1 = NUM SLASH
n2 = NUM {$indicacion = new IndicacionCompas($n1.int,$n2.int);};

constantes: constante*;

constante: CONST n1 = NOMBRE IGUAL (NUM {agregarConstante($n1.text, $NUM.int)}?
|n2 = NOMBRE{agregarConstante($n1.text, $n2.text)}?) PUNTOYCOMA;

melodia[IndicacionCompas indicacion] returns[List<Voz> voces] locals[int instrumento]
: {$voces = new ArrayList<Voz>();}
(VOZ LPAREN (NUM {$instrumento = $NUM.int;}|NOMBRE {constantes.containsKey($NOMBRE.text)}?
{$instrumento = constantes.get($NOMBRE.text);}) RPAREN
LBRACE compases[$indicacion] RBRACE {validarAlMenosUnCompas($compases.listaCompases)}?
{$voces.add(new Voz($instrumento, $compases.listaCompases));} )+ ;

compases[IndicacionCompas indicacion] returns[List<Compas> listaCompases] :
{$listaCompases = new ArrayList<Compas>();} compas[$indicacion] {$listaCompases.add($compas.compasObj);}
```

```
c1 = compases[$indicacion] { $listaCompases.addAll($c1.listaCompases);} |
{$listaCompases = new ArrayList<Compas>();} repeticion[$indicacion]
{agregarRepetidos($listaCompases,$repeticion.listaCompases,$repeticion.repeticiones);}
compases[$indicacion] | {$listaCompases = new ArrayList<Compas>();};

repeticion[IndicacionCompas indicacion] returns [List<Compas> listaCompases, int repeticiones]:
{$listaCompases = new ArrayList<Compas>();} REPETIR LPAREN NUM {$NUM.int > 0}?
RPAREN LBRACE (compas[$indicacion]
{$listaCompases.add($compas.compasObj);})+
{$repeticiones = $NUM.int;} RBRACE ;

compas[IndicacionCompas indicacion] returns[Compas compasObj]:
{$compasObj = new Compas();}COMPAS LBRACE (
nota {$compasObj.notas.add($nota.notaObj);} |
silencio {$compasObj.notas.add($silencio.silencioObj);}
)+RBRACE {validarDuracion($compasObj.notas, $indicacion)}?;

silencio returns[Nota silencioObj]: SILENCIO LPAREN DURACION PUNTILLO? RPAREN PUNTOYCOMA
{$silencioObj = new Nota(null,null,$DURACION.text,$PUNTILLO != null ? true : false);};

nota returns[Nota notaObj] : NOTA LPAREN ALTURA  COMA  octava  COMA DURACION PUNTILLO? RPAREN PUNTOYCOMA
{$notaObj = new Nota($ALTURA.text,$octava.valor,$DURACION.text, $PUNTILLO != null);};

octava returns[int valor]: OCTAVA {$valor = $OCTAVA.int;}| NOMBRE {$valor = constantes.get($NOMBRE.text);};
```

6.3. Codigo Midcomp

```
 package tleng.tp2;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

import org.antlr.v4.runtime.ANTLRFileStream;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.TokenStream;

import tleng.tp2.MusilengParser.Compas;
import tleng.tp2.MusilengParser.IndicacionCompas;
import tleng.tp2.MusilengParser.Nota;
import tleng.tp2.MusilengParser.NotaEnum;
import tleng.tp2.MusilengParser.Partitura;
import tleng.tp2.MusilengParser.Voz;
import static tleng.tp2.MusilengParser.clicksPorFigura;


public class midicomp {

  private static final String USAGE = "Usage : ./midicomp -c entrada.txt salida.midi";

  public static void main(String[] args) throws IOException {

    if (args.length != 3) {
      System.out.println(USAGE);
    } else {
      if (!args[0].equals("-c")) {
        System.out.println(USAGE);
      } else {
        String inFile = args[1];
        String midiFile = args[2];
        CharStream charStream = new ANTLRFileStream(inFile);
        MusilengLexer lexer = new MusilengLexer(charStream);
```

```
      TokenStream tokenStream = new CommonTokenStream(lexer);
      MusilengParser parser = new MusilengParser(tokenStream);
      Partitura partitura = parser.s().partitura;

      PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(midiFile)));

      //Encabezado
      int NTRACKS = partitura.voces.size()+1;
      double midi_tempo = 1000000 * 60 * NotaEnum.valueOf(partitura.tempo.duracion).getDuracion()
      / (4 * partitura.tempo.cantidad);
      String midi_bar = String.format("%s/%s", partitura.indicacion.tiempos,
      partitura.indicacion.tipoNota);
      writer.println(String.format("MFile 1 %d 384", NTRACKS));
      writer.println(String.format("MTrk"));
      writer.println(String.format("000:00:000 Tempo %.0f",midi_tempo));
      writer.println(String.format("000:00:000 TimeSig %s 24 8",midi_bar));
      writer.println("000:00:000 Meta TrkEnd");
      writer.println("TrkEnd");

      int i = 1;
      for (Voz voz : partitura.voces) {
        generarTrack(voz, i, partitura.indicacion, 384, writer);
        i = i + 1;
      }

      writer.flush();
      writer.close();
    }
  }

}

private static void generarTrack(Voz voz, int voice_number,
    IndicacionCompas indicacion, int clicks_per_beat, PrintWriter writer) {
  writer.println("MTrk");
  writer.println(String.format("000:00:000 Meta TrkName \"Voz %s\"",voice_number));
  writer.println(String.format("000:00:000 ProgCh ch=%d prog=%d",voice_number,voz.instrumento));
  int bar_num = 0;
  int beat_num = 0;
  int click_num = 0;

  for(Compas compas : voz.compases) {
    for(Nota nota : compas.notas) {
      if(!nota.isSilencio()) {
        writer.println(String.format("%03d:%02d:%03d On ch=%s note=%s vol=70",
        bar_num,beat_num, click_num,
        voice_number, nota.notacionAmericana()));
        int click_figure = clicksPorFigura(nota.duracion, indicacion);
        int temp_click = click_num + click_figure;
        click_num = temp_click % clicks_per_beat;
        int temp_beat = beat_num + (temp_click / clicks_per_beat);
        beat_num = temp_beat % indicacion.tiempos;
        bar_num = bar_num + (temp_beat / indicacion.tiempos);
        writer.println(String.format("%03d:%02d:%03d Off ch=%d note=%s vol=0",
        bar_num, beat_num,click_num,
        voice_number, nota.notacionAmericana()));
      } else {
        int click_figure = clicksPorFigura(nota.duracion, indicacion);
        int temp_click = click_num + click_figure;
        click_num = temp_click % clicks_per_beat;
        int temp_beat = beat_num + (temp_click / clicks_per_beat);
        beat_num = temp_beat % indicacion.tiempos;
        bar_num = bar_num + (temp_beat / indicacion.tiempos);
      }
```

```
      }
    }
    writer.println(String.format("%03d:%02d:%03d Meta TrkEnd",bar_num, beat_num, click_num));
    writer.println("TrkEnd");
  }

}
```