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	FALTA LU	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

- En primera instancia definiremos una gramática necesaria para interpretar, de manera correcta, nuestro archivo de entrada.
- Luego definiremos cada expresion regular como tokens en el lexer_rules, de esta manera cargamos en la computadora nuestras expresiones regulares definidas en la gramática.
- El siguiente paso será definir las producciones correspondiente, las cuales gracias a los tokens definidos en el paso anterior, podremos construir, diferenciarlas una de las otras y filtrar aquellas que no sean válidas.
- Cada producción llamará a su función interna, formando el árbol decado desde las hojas hasta su raiz, usando en cada una atributos sintetizados para poder intercambiar valores de una rama a la otra y poder validar las condiciones especificadas en nuestro lenguaje.
- Una vez corroborado y habiendo obtenido con éxito todo lo anterior, procederemos a escribir el archivo MIDI cumpliendo las normas del mismo sin dificultades.

2. Descripción del problema resuelto

En esta sección explicaremos cada parte implementada para realizar el tp. Contaremos dudas, errores que fueron surgiendo y explicaremos las decisiones tomadas. Para eso vamos a dividirlo en cuatro secciones que detallamos a continuación.

2.1. Errores surgidos durante la implementación

2.2. Lexer

2.3. Parser

2.4. Midi

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}

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 = 10;
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(mi, oct2, negra .);
45      nota(fa, oct4, corchea);
46      nota(mi, oct2, negra);
47    }
48    compas
49    {
50       silencio (semifusa);
51      nota(si+, oct1, semifusa);
52      nota(si, oct4, semifusa);
53      nota(fa, oct4, semifusa);
54      nota(sol, oct4, semicorchea);
55      nota(mi-, oct4, corchea);
56      nota(re, oct4, negra);
57    }
58  }
```

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.

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.