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 caputar. 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 "sol+.[en] una nota, buscara una regla que inicialice con un + o - y no la encontrará lógicamente. 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.

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 = 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)
{
```

```
29    compas
30    {
31      nota(do, oct3, fusa);
32      nota(re, oct1, semifusa.);
33    }
34
35    compas
36    {
37      nota(mi, oct2, fusa);
38      nota(la, oct1, semifusa.);
39    }
40  }
```

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

```
1  #tempo negra 60
2  #compas 1/1
3
4  const oct1 = 3;
5  const oct2 = ConstanteTrucha;
6
7  // Instrumentos
8  const bajo = 20;
9
10 voz (bajo)
11 {
12   compas
13   {
14     nota(do, oct1, blanca.);
15     nota(re, oct1, negra);
16   }
17
18   compas
19   {
20     nota(mi, oct2, blanca.);
21     nota(la, oct2, negra);
22   }
23 }
```

4.1.4. Test 4: Constante definida circularmente

```
1  #tempo negra 60
2  #compas 2/8
3
4  const oct1 = 3;
5  const oct2 = 5;
6
7  // Instrumentos
8  const bajo = 20;
9  const malPensado = malPensado;
10
11 voz (bajo)
12 {
13   compas
14   {
15     nota(do, oct1, corchea.);
16     nota(re, oct1, semicorchea);
17   }
18
19   compas
20   {
```

```
    nota(mi, oct2, semicorchea);
    nota(la, oct2, corchea.);
  }
}
```

4.2. Tests correctos

4.2.1. Test 1: Simple

```
#tempo negra 30
#compas 2/4

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

// Instrumentos
const flauta = 51;

voz (flauta)
{
  compas
  {
    nota(do, oct3, negra);
    nota(re, oct1, negra);
  }

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

4.2.2. Test 2: Con varias voces

```
#tempo negra 120
#compas 3/4

const oct1 = 2;
const oct2 = 6;
const oct3 = 1;
const oct4 = 3

// Instrumentos
const piano = 65;
const violin = 31;

voz (piano)
{
  compas
  {
    nota(sol, oct3, blanca);
    nota(fa+, oct2, negra);
  }

  compas
  {
    nota(mi, oct2, negra.);
    nota(fa, oct4, corchea);
    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.
- 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 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.