Teoría de Lenguajes

Primer Cuatrimestre de 2015

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

Trabajo Práctico 2

Grupo Estado Final

Integrante	LU	Correo electrónico
Gorojovsky, Román	530/02	rgorojovsky@gmail.com
Lazzaro, Leonardo	147/05	lazzaroleonardo@gmail.com

Introducción

El trabajo consiste en implementar un programa que tome un archivo escrito en el lenguaje Musilen, definido por la cátedra, y convertirlo en un archivo midi, pasando por un lenguaje intermedio que es interpretado por el programa *midicomp* para generar finalmente el archivo midi.

El problema, entonces puede dividirse en tres subproblemas:

- Convertir la descripción del lenguaje Musilen en una gramática bien definida.
- Aprender a usar alguna biblioteca preexistente para convertir esa gramática en código
- Convertir la salida del *parser* creado en los dos pasos anteriores en el lenguaje de *midicomp*

Decisiones tomadas

Se eligió usar *PLY* debido a nuestro mejor manejo del lenguaje *Python* y se usó como esqueleto del trabajo el código presentado en clase:

- `lexer_rules.py` contiene las definiciones y reglas de tokens
- `lexer.py` es el código de ejecución del *lexer* (usado principalmente para testeo.
- `parser_rules` contiene la definición de la gramática
- `expressions.py` contiene los objetos que se crean a partir del análisis sintáctico.
- `parser.py` es el código de ejecución del *parser*

Como se verá en la sección que detalla la implementación, la conversión de objetos al lenguaje *midicomp* está implementada dentro de los objetos de `expressions.py`, usando patrones de programación orientada a objetos.

Gramática

Se define la siguiente gramática, para cuya definición se priorizó la simplicidad en la construcción de los objetos que luego se usarán para generar el archivo *midicomp* por sobre la legibilidad de la gramática. En general se pusieron los terminales (puntos y coma, llaves de cierre) en las producciones "de más afuera".

Notamos en negrita los tokens y en negrita bastardilla los tokens con algún valor, definidos más abajo.

Musilen → DefTempo DefCompas Constantes Voces
DefTempo → **#tempo** Duracion ***num***
DefCompas → **#compas** ***num/num***
Constantes → λ | constante **;** constantes
Constante → **const** ***constante*** **=** ***num***
Voces → Voz **}** | Voz **}** Voces
Voz → **Voz** (Var) **}** ListaCompases
ListaCompases → ListaCompases CompORepe
CopmORepe → Compases | Repetir
Repetir → **repetir** (***num***) **{** Compases **}**
Compases → Compas **}** | Compas **}** Compases
Compas → **compas {** Figuras
Figuras → Figura **;** | Figura **;** Figuras
Figura → Nota | Silencio
Nota → **nota** (***altura***, Var, ***duracion***)
Silencio → **silencio** (***duracion***)
Var → ***constante***| ***num***

Los tokens con valor son:

num = 0|[1-9][0-9]*
duracion = (redonda|blanca|negra|corchea|semicorchea|fusa|semifusa)(.)?
altura = (do|re|mi|fa|sol|la|si)(-|+)?
constante = [a-zA-Z]+

La gramática que se implementa en `parser_rules.py` tiene una pequeña diferencia con esta: no existe el no terminal "ComoORepe", que se agregó acá para simplificar la lectura, pero está implementado directamente en la lista de compases. Además, si bien

Implementación

Uso del programa

Tests y ejemplos