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

TP2 - Musileng

Teoría de Lenguajes - TP2

GRUPO Los Sin Grupo

Juan Enríquez - LU 36/08 - juanenriquez@gmail.com
Damián Furman - LU 936/11 - damian.a.furman@gmail.com

Resumen

En el presente informe mostramos como se llevó a cabo la implementación de un lexer y un parser para el lenguaje musileng el cual nos permite una escritura amigable de archivos midi a través de la generación de un archivo intermedio que además se genera por nuestro programa.

Keywords

Lenguaje musical - Musileng - MIDI - Parsing - LALR - PLY

1. Introduccion

Dentro del marco de el reconocimiento y la generación de lenguajes vimos un número de técnicas que nos permiten estudiar la estructura sintáctica de un lenguaje. Estas técnicas, también llamada *parsing* nos permiten comprender un lenguaje desde su estructura a partir de una gramática. Es decir, podemos chequear (o incluso generar) un lenguaje a partir de un conjunto acotado de reglas que nos dicen como se construye un lenguaje.

En nuestro caso, se nos pidió realizar un programa que pueda ser capaz de *reconocer* un lenguaje llamado *Musileng*.

Musileng es un lenguaje que nos permite realizar composiciones musicales en formato MIDI a través de una especificación dada que luego será convertida a un formato intermedio para ser a su vez procesado oportunamente por otro programa que se encargará de generar el archivo de audio en el formato antes mencionado.

2. Gramática

Parte del trabajo necesario para poder cumplir con la tarea asignada era poder definir la gramática del lenguaje Musileng. En nuestro caso, si bien contamos con una especificación del lenguaje y como eran sus estructuras de control no contamos con la definición formal de la gramática del mismo motivo por el cual procedimos a definirla nosotros mismos. A continuación se detalla la gramática que se dedujo a partir de la especificación brindada.

Sea G una gramática libre de contexto tal que $G :< V_n, V_t, P, S >$ con P definido como:

S → ENCABEZADO CONSTANTES VOCES
ENCABEZADO → TEMPO COMPAS_DEF
TEMPO → #FIGURA numero
FIGURA → blanca | redonda | negra | corchea | semicorchea | fusa | semifusa
COMPAS_DEF → #compas numero/numero
CONSTANTES → const cadena numero; CONSTANTES | λ
VOCES → DECLA_INST { MUSICA }
DECLA_INST → voz(numero)
MUSICA → COMPAS MUSICA | BUCLE MUSICA | λ
COMPAS → compas { NOTAS }
BUCLE → repetir(numero) { COMPAS COMPASES }
COMPASES → COMPAS COMPASES | λ
NOTAS → FIGURA NOTAS | λ
FIGURA → NOTA | SILENCIO
NOTA → nota(ALTURA, numero, DURACION)
ALTURA → string SIMBOLO
SIMBOLO → + | − | λ
DURACION → string | string •
SILENCIO → silencio(DURACION)

3. Implementación

Una vez que contamos con la gramática definida para el lenguaje en cuestión teníamos dos puntos fundamentales para resolver.

Por un lado implementar un parser para poder comprender la estructura sintáctica de nuestro lenguaje y, por otro lado, era necesario poder darle semántica a lo que estabamos tratando de comprender para poder generar ciertas estructuras y recavar cierta información necesaria de lo que se leía para poder generar un archivo intermedio en otro lenguaje para poder, efectivamente, componer música a través un programa externo.

Respecto de la primera tarea lo que hicimos fue utilizar por recomendación de la cátedra un *generador de parser y lexer* llamado PLY. PLY es un envoltorio para el lenguaje de programación Python de históricos lexers y parser de Unix llamados `lex` y `yacc`.

Lex es un analizador léxico. Es un programa cuya tarea es tomar una secuencia de caracteres y retornar una secuencia de *tokens*, es decir, una secuencia de símbolos pero que tienen un sentido particular. Por su parte, yacc es un generador de parsers que nos permite a partir de un conjunto de reglas (ie. la definición de una gramática) generar un parser de tipo LALR que reconoce el lenguaje descripto por las reglas mencionadas.

Para el lexer de nuestro lenguaje se procedió con la implementación de un conjunto de reglas que lo que hacen es partir el archivo de entrada en tokens con un significado relevante para el contexto de nuestro programa. En nuestro caso, queremos que los strings `do`, `re` o `sol` no sean simplemente esos strings sino que sean tokens que reflejen cierto comportamiento común de acuerdo al contexto de uso, en particular para nuestro programa son `NOTAID` que es un token que representa la ocurrencia de una nota musical dentro del programa. Lo mismo ocurre con cosas como `const grand_piano = 1;`. Esto no representa cualquier cosa sino que simboliza la definición de una constante con una *keyword* particular y un nombre con un valor y otros símbolos que permiten identificar esta construcción sintáctica.

En una etapa posterior se hizo algo similar con el parser. Yacc nos pide definir las reglas de la gramática necesaria para operar sintácticamente con el lenguaje a estudiar. Estas reglas se corresponden con la gramática definida en la sección anterior pero están definidas a partir de los tokens que se establecieron en el primer paso. Adicionalmente, las reglas que se definen permiten agregarle comportamiento al parser y poder así escribir el archivo de salida y así también poder validar ciertas cuestiones relacionadas con la composición musical como respetar la duración de los compases y validar su estructura.

Además de recolectar información sobre la estructura de la composición durante el parseo se programaron estructuras adicionales que facilitan la conversión al formato intermedio de manera de poder realizar esta tarea de manera lo más cómoda posible.

4. Conclusiones

A modo de conclusión del trabajo hay dos puntos que nos resultan importante mencionar. Por un lado, creemos que el trabajo, si bien tiene una componente algorítmica o general de programación que es independiente al contenido de la materia, es un buen exponente de casi todos los temas que se han visto a lo largo de la cursada. Más aún, es una instanciación completamente práctica de una materia con una fuerte carga teórica y creemos que este balance es positivo porque nos permite atacar los temas desde dos frentes muy importantes.

Por otro lado, nos pusimos a pensar que hubiéramos hecho si este trabajo hubiera sido parte de otra asignatura previa a Teoría de Lenguajes. Creemos que la resolución del mismo no hubiera sido posible (o al menos de una forma elegante) sin tener las herramientas tanto teóricas como prácticas para alcanzar este objetivo. No sólo que estaríamos reinventando la rueda sino que además creemos que la solución alcanzada por fuera de este marco sería algo completamente poco robusta y con un alto costo de modificación o extensión.

Esto no implica que la tarea de leer un archivo, interpretarlo y generar otro que un significado parecido sea algo trivial. Más bien todo lo contrario, creemos que es una tarea bastante compleja a pesar de los avances con los que contamos y que vimos a lo largo de la materia. Especialmente compleja cuando hablamos de tareas que se agregan en este proceso como por ejemplo la optimización del código que se genera y demás cosas que son importantes en el diseño de compiladores.