**Teoría de Lenguajes y Automatas**

Trabajo Practico Especial: Conversor de Gramáticas – Automatas Finitos

Alumnos:

Federico Di Nucci, legajo 50120

Agustín Marseillan, legajo 50134

Nicolás Loreti, legajo 49479

Índice

Introducción y Objetivo 3

Consideraciones Realizadas 4

Desarrollo del Trabajo Práctico 6

Problemas durante el Desarrollo 8

Posibles Extensiones 9

Introducción y Objetivo

El trabajo consiste en generar un conversor de gramáticas regulares a autómatas finitos y viceversa. Dada una gramática regular se debe obtener una representación grafica del autómata finito equivalente. Dada una especificación del autómata finito, debe poder obtener la especificación de la gramática equivalente.

Consideraciones Realizadas

**DOT:** Dependiendo de la plataforma el programa Dot se encuentra en diferentes rutas, por ejemplo en Ubuntu está en “/usr/bin/dot” mientras que en otra plataforma como MacOS el resultado de ejecutar un which dot da como resultado el path “usr/bin/local/dot”. Esto trae severos problemas de portabilidad del programa ya que no permite que un mismo ejecutable sin pasarle nada más que la gramática encuentre la ruta donde se ubica el programa Dot y realice todo de manera correcta.

Para solucionar esto decidimos pasarle como parametro al programa la plataforma desde donde se está ejecutando dando como opciones los antes mencionados Ubuntu y MacOS. A partir de esta variable el programa podrá identificar la ruta donde se encuentre instalado por defecto el archivo necesario para hacer las visualizaciones PNG de las gramáticas asociadas.

**Linux:** El programa debe ser corrido en alguna plataforma Linux ya que hace uso de las directivas fork() y execv(), estas fueron necesarias para no hacer muy engorroso el código lex, ya que ahora los dos arichivos .l se encuentran separados. También fue necesario al momento de crear la imagen con el Graphviz.

**Espacios Vacíos:** Ya que en ningun momento se especifica como tratar los espacios vacios dentro de las gramáticas decidimos representarlos mediante un 0. Es decir que la producción A->c quedaría representada por A->c0 dentro del programa.

**Archivos de Salida:** No se realiza un nuevo archivo por cada ejecución sino que la salida se da siempre en los mismos archivos que se van sobreescribiendo para dar lugar a los nuevos. Esto hace que si el usuario quiere ir guardando distintas salidas tenga que ir copiando los resultados a una carpeta personal para que estos no se pisen con la próxima ejecución del conversor.

Descripción del desarrollo del TP

El desarrollo estuvo separado basicamente en dos etapas, o fases. La primera correspondió a el parseo utilizando Lex tanto de la Gramática como del Autómata. Para esto fue necesario aplicar los conocimientos adquiridos en clase durante la Práctica 6. El uso de etiquetas resultó fundamental para poder armar una especie de maquina de estados que permita moverse entre uno y otro a lo largo del parseo de las distintas cadenas.

Se creó un archivo Lex para cada uno de los conversores propuestos por la cátedra. Es decir se creó un Gramatica.l para el parseo de las gramáticas y un Automata.l para hacer lo propio con los autómatas. Además cada uno de ellos tenían en la última zona de código el programa *Main()* y el resto de las funciones necesarias para la implementación de la salida.

La segunda etapa consistió justamente en la implementación de todo lo relacionado con la interpretación de lo parseado y el armado de una salida que corresponda con lo planteado en el TPE. Para esto contamos con una estructura structs.h en la que guardamos y definimos todas las estructuras necesarias para modelar tanto las gramáticas como los autómatas.

Armar un buen modelado fue clave en el desarrollo ya que eso nos simplifico la manera en gran medida y las acciones que teníamos que llevar a cabo resultaban sencillas de interpretar. Para ser más precisos en el caso de las Gramáticas se creo una estructura para los símbolos no terminales, para los terminales, para el símbolo inicial y luego una estructura más compleja para modelar las producciones.

Una vez que el parser de Lex cargaba toda la información en las estructuras correspondientes desde el programa C modificamos la entrada para que cumpla con los requisitos planteados. Para esto tuvimos que realizar diversas acciones como: Eliminar unitarias, Eliminar inalcanzables, Eliminar improductivos, determinar cuando una Gramática o Autómata estaba mal formado y devolver un mensaje de error en consecuencia, etc.

Luego se imprimían todos los resultados necesarios por la salidad estandar especificando con un título a que correspondía la información que el usuario veía en la pantalla.

Una vez que estabamos seguros de que teniamos una entidad válida que cumplía con los requisitos planteados procediamos a armar la salida, ya sea el .dot para que se pueda generar la imagen png del autómata correspondiente a la gramática ingresada por el usuario o viceversa.

Esto último termina la ejecución del programa ya que todos los pasos fueron llevados a cabo.

Dificultades Encontradas

En cuanto a dificultades que se nos hayan encontrado en el ambito de lex, una de las principales fue la de rechazar ciertos patrones que se daban en los archivos malformados, el parseo para los que estaban bien escritos fue mucho mas sensillo.

Dentro de lo que fue el desarrollo en C la mayor dificultad estuvo en pasar los algorítmos vistos en clase y hechos en papel a algoritmos computadorizados. Si bien los algoritmos son sencillos y facilmente implementables en la realidad, a la hora de usar C y manejar punteros hubo que tener mucho cuidado con los valores que uno coloca en las producciones mientras las iba a modificando para poder obtener la salida esperada.

En cuanto al pasaje de Gramática a Autómata el desafío más importante se encontró en eliminar las producciones inalcanzables, improductivas y por último normalizar la gramática para que pueda ser exportada a un archivo .dot.

También el hecho de estar dependiendo de un programa externo como Dot que se instala en diversos directorios dependiendo de la plataforma no facilitan la portabilidad del conversor y si bien no fue una dificultad mayor genera complicaciones si se quiere escribir un programa multiplataforma.

De habérsenos dado la opción del lenguaje hubiésemos preferido usar Java ya que mediante el manejo de objetos y clases, getters y setters todo el desarrollo hubiese sido más ameno y se hubiese realizado en menor cantidad de tiempo.

Futuras Extensiones