**Teoría de Lenguajes y Autómatas**

Trabajo Practico Especial: Analizador Sintáctico Descendente con Retroceso

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 5

Problemas durante el Desarrollo 6

Posibles Extensiones 7

Introducción y Objetivo

El trabajo consiste en escribir un programa el cual dado una gramática libre de contexto, genere el analizador sintáctico descendente para dicha gramática en lenguaje C, el mismo deberá recibir una cadena como argumento y decidir si la misma es generada con la gramática usada para crear el analizador y además dar las producciones utilizadas para generar dicha cadena.

Consideraciones Realizadas

**Gramáticas:** Suponemos que los símbolos no terminales de la gramática son letras mayúsculas, y que los terminales son minúsculas. También asumimos que todos los símbolos que se encuentran en las producciones son símbolos validos, es decir que fueron declarados como terminales o no terminales.

**Espacios Vacíos:** Ya que en ningún momento se especifica como tratar los espacios vacíos 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 sobrescribiendo 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, al igual que en el primer trabajo practico especial, estuvo separado básicamente en dos etapas. La primera correspondió a el parseo utilizando Lex , para esto se pudo utilizar el parser del trabajo anterior con algunas modificaciones, ya que ahora se deberá aceptar que del lado derecho de una gramática pueda haber mas de dos símbolos. Al haber parseado la gramática y el autómata en diferentes archivos Lex en el trabajo anterior, se pudo adaptar muy fácilmente a este. Gracias a esto se pudo reutilizar las mismas estructuras también.

La segunda etapa consistió en la implementación del programa el cual será el encargado de escribir el analizador sintáctico, para ellos se utilizo los algoritmos vistos en la teórica. Se creo un procedimiento por cada símbolo no terminal los cuales brindaban sus producciones a la función procesar que se encarga del rechazo o aceptación de la producción , una función main que llamaba al procedimiento del símbolo inicial y luego se verificaba si había salido con error o si no se había consumido toda la cadena, en caso que no fuese así, la cadena seria aceptada.

Dificultades Encontradas

Dentro de lo que fue el desarrollo en C la mayor dificultad estuvo en pasar los algoritmos vistos en clase y hechos en papel a algoritmos computadorizados. Si bien los algoritmos son sencillos y fácilmente 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.

Un gran desafío con el cual nos enfrentamos fue la implementación de un programa el cual a su vez escriba otros programas. Hubo que pensar bien que cosas había que hacer. La detección de errores fue mas complicada, ya que había que verificar dentro de que archivo se encontraba el error.

Otra barrera con la cual nos chocamos fueron las transiciones lambda, al principio fue un gran dolor de cabeza tratar con las mismas, ya que no sabíamos como manejarlas.

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

Como futura implementación, se podría considerar la opción de eliminar repetidos, recursividad a izquierda, transiciones Lambda para así optimizar el algoritmo. También se podría ,utilizando mismo parser y estructuras, implementar un analizador sintáctico ascendente.