**Trabajo Práctico No2**

**Teoría de Lenguajes y Autómatas**

**Integrantes:**

**Homovc, Federico**

**Reznik, Luciana**

**Pereyra, Cristian**

**1er cuatrimestre, año 2012**

**Índice**

**Objetivo del Trabajo Práctico**

El trabajo consiste en programar en C un Generador de Analizador Sintáctico Descendente con Retroceso(ASDR). Dada una gramática libre de contexto, sin recursividad a izquierda, el programa deberá generar un ASDR que estará escrito en lenguaje C.(ASDR.c). El ASDR podrá luego recibir una cadena de entrada y determinar si pertenece o no al lenguaje generado por la gramática, mostrando además la derivación que llevó a dicha cadena.

**Consideraciones realizadas**

Para hacer el trabajo práctico se buscó centrarse en lo visto en las clases teóricas de la materia, y se trató de realizar la programación con los algoritmos vistos en las prácticas.

**Descripción del desarrollo del TP**

Para la realización del trabajo, se utilizaron las librerías de la materia Sistemas Operativos, para facilitar el parseo de las estructuras de autómata y gramática. Utilizando listas, mapas y conjuntos, la lógica de las implementaciones se simplificó bastante y se redujo a un nivel más comprensible y legible.

El programa comienza en archivo tp1.c. El mismo lee el archivo que se le pasa por parámetos y en caso de ser un .gr realiza el paso de gramática a autómata y si es un .dot llama a la función que pasa de autómata a gramática.

Los valores para las estructuras se obtienen parseando el archivo de entrada mediante expresiones regulares de Lex.

**Estructuras principales utilizadas para el TP**

**● Grammar**

La *gramática* está conformada por una lista de *símbolos terminales,* una de *símbolos no terminales,* el *símbolo inicial* y un mapa con las producciones. Las *producciones* contienen a su vez un string inicial y una lista de símbolos de la producción.

Además posee las funciones para el manejo de la gramática (determinar si es una gramática regular, si es válida, si se puede pasar a regular, etc.), la función para pasar a la forma derecha y la función *grammar\_to\_automatha* que pasa la gramática a autómata.

**Función grammar\_to\_automatha:**

Para pasar a autómata, antes que nada, se remueven las producciones unitarias, inalcanzables e improductivas. Luego la función chequea que pueda ser pasada a gramática regular. Si no se puede, devuelve NULL, en caso contrario, y si es de tipo izquierda la convierte en derecha.

Para esto, se agrega el simbolo no terminal M y la transicion M->lambda, y por cada no terminal en una produccion un simbolo no terminal y una transicion a ese no terminal y el simbolo M. Por ejemplo en caso de tener una produccion A->b, se agrega B->bM y A->B ademas del simbolo B en el conjunto de no terminales.

Luego, se transforman todas las producciones del tipo A->Cb en C->bA y para el caso de las lambda se marca el simbolo como nodo terminal.

Finalmente, se imprimen los nodos con sus transiciones en un archivo .dot y se llama al programa Graphviz con dicho archivo para generar la imagen correspondiente.

**● Automata**

El *autómata* está conformado por un mapa que contiene los distintos estados del mismo.

Un *estado* es otra estructura que está conformada por un nombre, una etiqueta, una lista con las transiciones posibles y un flag que representa si es o no un estado final.

Las *transiciones* contienen el *destino* y un *token* que es el símbolo terminal que la cadena debe consumir para pasar al siguiente estado.

Además posee las funciones para el manejo del autómata (agregar nodos, transiciones, etc.) y la función *automatha\_to\_grammar* que hace el paso de autómata a gramática.

**Función automatha\_to\_grammar:**

Al principio del procedimiento, se cargan los datos parseando el archivo de entrada y se verifica que el grafo de entrada sea conexo. En caso contrario, se imprime un mensaje de error y no se prosigue con el algoritmo.

Si el archivo de entrada es valido, se crea una lista de los estados del autómata así como un mapa para las producciones de la gramática, un árbol para los símbolos terminales y otro para los no terminales. Luego, por cada estado del autómata, se agrega a la lista de símbolos no terminales el estado (si es el inicial se marca como tal) y por cada transición de ese estado se genera la producción correspondiente o se agrega a la lista de producciones si ya existia. En cada transicion, por cada simbolo terminal se lo agrega al conjunto de simbolos terminales.

**Dificultades encontradas en el desarrollo del TP**

Dado que hacía un tiempo que no programábamos en el lenguaje C, algunas de nuestras librerías no funcionaban como lo esperado (la librería de cadenas y el mapa).

Tuvimos varias confusiones a la hora de tener que trabajar con gramáticas de forma derecha o izquierda, y verificar si se puede normalizar o no una gramática.

Otro problema que se nos presentó, fue la portabilidad en Linux. Lo probamos en Macintosh y andaba pero en Linux no compilaba por lo que tuvimos que adaptar el programa para que fuera lo más portable posible.

En un momento, una función accedía ilegalmente a memoria sin que pudiéramos saber por qué. Finalmente pudimos deducir que se debía a que intentábamos modificar una cadena del segmento de datos. Esto llevó varias horas de debugging hasta encontrar la causa.

Utilizar directamente graphviz también fue una complicación. Debido a ello, decidimos utilizar la llamada "system" y ejecutar directamente dot con los parámetros necesarios para generar el png. El programa no generaría imágenes si graphviz no se encuentra instalado en el sistema.

**Futuras extensiones**

Sin duda, una de las extensiones que deberíamos realizar a futuro, es la minimización de los autómatas ya que es algo muy útil e interesante a la hora de pasar de gramática autómata. La implementación de tal extensión puede ser pesada pero tenemos distintas funciones que podrían ayudar (como la de sacar las transiciones lambda).

Otra extensión que podríamos implementar, es la de intentar llevar la gramática a forma normal. Si bien lo hacemos, el método es bastante precario, y podría ser más potente. En caso de que la gramática no fuera regular, sería una pérdida de tiempo.