Gramáticas Regulares y Autómatas

Mesa Alcorta, María Victoria
Pierri, Alan
Rey, Juan Pablo

October 6, 2012

Contents

1 Consideraciones realizadas

La forma de compilar y correr el programa está en el archivo “README.md”.

1.1 Gramáticas Regulares a Autómatas

Dentro del trabajo práctico tuvimos que suponer algunas cosas debido a que el enunciado, naturalmente, da lugar a ello.

En primer lugar, para el nombre de la gramática, tuvimos que asumir que cuando el enunciado nos pide que el nombre de la gramática sea una letra seguido de cualquier caracter (sin espacios), supusimos que cualquier caracter era válido hasta un espacio o tabulación o salto de línea. Nos pareció incoherente que un nombre de una gramática posea tabulaciones y saltos de línea. Es decir, consideramos que el nombre de la gramática podria tener, luego de la primera letra, cualquier letra, numero o metacaracter. Eventualmente, el nombre de la gramática podria ser "G123o{}{}+_-=_@$#%" o "G1={A,B,C}", por eso uno debe tener la precaución de espaciar/tabular/saltar de línea finalizado el nombre.

En cualquier otro caso, supusimos que entre cualquier par de caracteres (obviando el nombre o las "flechas") pueden haber cualquier cantidad de espacios/tabulaciones/saltos de línea.

Cualquier caracter luego de la gramática que no sea espacio/tabulación/salto de línea da ERROR.

Un terminal o no terminal podría no usarse en las producciones.

Para que sea válida, todos los no terminales y terminales de la producción están declaradas.

Además de imprimir el nombre de terminales y no terminales por salida, imprimimos las producciones. Nos pareció apropiado a pesar de no solicitarlo en el enunciado.

Al pasar de una gramática regular a un autómata, en muchos casos hace falta un extra nodo para normalizar las producciones. Como los no terminales son letras y debería aceptar una gramática con todas las letras, nombramos a este nodo con el caracter ’0’.

El nombre de la gramática creada luego de normalizar y remover improductivos e inalcanzables se llama para nosotros "Nueva Gramática".

Como el nombre de la gramática puede tener cualquier secuenta de metacaracteres y eso imposibilitaría que el archivo tome cualquier nombre que la gramática posea, decidimos que el archivo .dot y la imagen .png creada se llamarían "out.dot" y "out.png" respectivamente.

Como no especificaba si era de la gramática recibida o de la gramática generada que se tenia que imprimir por pantalla los datos, imprimimos los datos de ambas gramáticas.

1.2 Autómatas a Gramáticas Regulares

Se implementó como pedía el trabajo práctico, la conversión de un Autómata finito determinístico a una gramática regular.

Para ello se utilizó el algoritmo provisto por la cátedra (Pasaje de AFD a GR. Parte 4).

Nuestro nodo inicial empieza con Node0 en vez de el que tenga el label 0. Esto fue debido a un error de interpretación.

Para esta parte, se realizan múltiples validaciones, conforme se va parseando. Un archivo válido .dot tiene la forma:

Digraph{
// Nodos
node [shape=circle] Node0 [label="0"]
//Transiciones
Node0->Node0 [label="b"];

Por empezar se realizan todas validaciones sintácticas por "lex".

En el caso en el que se agregue un estado inicial, se fija que no haya otro ya, si ya hay otro tira error (un autómata debe tener un solo estado inicial).

Cuando se agrega una transición, se fija que el estado "To" de esa transición haya sido declarado antes, sino tira error, en el caso que sí haya sido declarado, lo marca. Cuando se termina de cargar la última transición, se termina de armar la estructura "Automaton" y se llama a la función "validateAutomaton". Esta función se fija que el autómata tenga un solo estado incial y que sea conexo.

Una vez que se tiene una estructura "Automaton" cargada con un autómata válido. Se llama a la función "fromAFDtoGR" que pide memoria y carga la gramática equivalente. Y a partir de esa gramatica se crea el .gr con la misma.

2 Desarrollo del Trabajo Práctico

En principio nuestro programa no tiene parámetros. Así que uno puede enviarle tanto un automata (en formato .dot) como una gramática.

2.1 Gramáticas Regulares a Autómatas

Para desarrollar esta parte, creamos una serie de estados que nos permite orientarnos y saber que estamos esperando. Esta nos pareció la forma mas sencilla de analizar un *string* sin tener en cuenta el contexto. De esta manera si no cumple con ninguna expresión dado un estado, el programa notifica que el formato de entrada es incorrecto. Luego de haber finalizado con éxito la interpretación de la gramática, se valida. Un proceso más de análisis es pasar a gramática normal derecha, lo cual consta de dos partes:

Primero, una normalización de la gramática, sin importar la orientación. Esto permite no solo que los datos sean muchisimo mas manipulables sino que es un requisito de la consigna que ya se descarta, de ser de derecha.

Luego, analiza si es de izquierda. Si lo és, se lleva a una forma normal derecha mediante el algoritmo utilizado en clase.

Finalmente, se remueven todos los nodos improductivos e inalcanzables, y así se convierte toda la gramática en un .dot que, por una línea de comandos (dot -Tpng out.dot > out.png) se convierte en una imágen PNG.

2.2 Autómatas a Gramáticas Regulares

Para el desarrollo de esta parte, se crearon estructuras que permitieran extender el trabajo a la conversión de un AFND-Lamda a gramática.

Para eso, en la estructura que describe una transición, "transition" , en lugar de poner un char en el campo "to" se dejó una lista, que en este trabajo solo tendrá un único elemento, pero en caso de extenderlo a otro tipo de autómatas se podrá hacerlo fácilmente.

Por empezar, se parsea el archivo .dot y se valida el autómata recibido, luego se cargan las estructuras que describan a ese autómata y se llama a la función "fromDot" que implementa el algoritmo "Pasaje AFD a GR" esta función pide la memoria necesaria para la grámatica equivalente al autómata, y la inicializa.

A partir de esa gramática, se llama a la función "GrammarToString" que convierte esa gramática a un char * para crear el archivo válido .gr.

3 Futuras extensiones

Como se explico en "2.2 Automatas Finitos Determinísticos a Gramatica", se dejaron las estructuras listas de modo que el programa pueda recibir Autómatas Finitos No Determinísticos Lambda. Solo quedaría implementar los algoritmos para pasar de AFND-Lambda a AFND, luego de AFND a AFD y a partir de ahí llamar a la función ya implementada que pasa de AFD a gramática.