

22 de noviembre 2017

***Instituto Politécnico Nacional***

**Escuela Superior de Cómputo (ESCOM)**

***TAREA 2: INVESTIGACIÓN DE CONCEPTOS***

Unidad de Aprendizaje: Teoría Computacional

Profesor: Rafael Aguilar García

Alumno:

* Ramos Diaz Enrique

Grupo: 2CM2

**1.- Teorema de Myhill-Nerode**

Para comprobar que un lenguaje independiente de contexto es o no regular existen dos métodos muy útiles: el lema de Bombeo y el Teorema de Myhill-Nerode.

De manera formal, el teorema dice que debemos de tener un lenguaje L como subconjunto del universo Ʃ\*, para que de esta forma sean equivalentes las siguientes afirmaciones:

* L es aceptado por un AFD o un AFND
* L es la union de las clases de equivalencia de una relacion de equivalencia en Ʃ\* de indice finito, invariante por la derecha
* La relacion de equivalencia RL es de indice finito

Las clases de equivalencia son todos los posibles casos que una palabra del alfabeto puede cumplir segun la condicion del lenguaje. En otras palabras, son los estados de algun automata finito. La unión de éstas forman todas las palabras que el alfabeto puede generar (el universo).

Así, aclarado esto, basicamente el teorema nos dice que, si existen infinitos casos que una condicion de algun lenguaje podria arrojar, automaticamente este lenguaje no es regular, pues contradice al número finito de estados de algun automata.

Para comprender mejor el teorema, utilizaremos dos ejemplos de lenguajes: uno regular y otro no regular, vistos en clase:

**Ejemplo 1: L = { w | |w|a = |w|b = |w|c }, es decir, el lenguaje que posea palabras con la misma cantidad de ‘a’s, ‘b’s y ‘c’s. En Ʃ = {a, b, c}**

Para denotar este comportamiento, utilizaremos la siguiente expresion: anbncn, donde n es mayor a cero. Al escribir algunas clases de equivalencia:

* Si n = 1, entonces [abc] = Todas aquellas palabras con una sola a, b y c. **Ej: abc.**
* Si n = 2, entonces [a2b2c2] = Todas aquellas palabras con dos a’s, b’s y c’s. **Ej: aabbcc.**


* Si n = k, entonces [akbkck] = Todas aquellas palabras con k-numero de a’s, b’s y c’s. **Es decir, las a’s, b’s y c’s son infinitas. Ej: aaaa....bbbb....ccccc.....**

Y este solamente es el caso cuando a, b y c son iguales, aun faltan todas las demás posibles combinaciones donde alguna vaya variando y las otras se queden constantes. El universo será Ʃ\* = [abc] U [a2b2c2] U ... U [akbkck] U ...

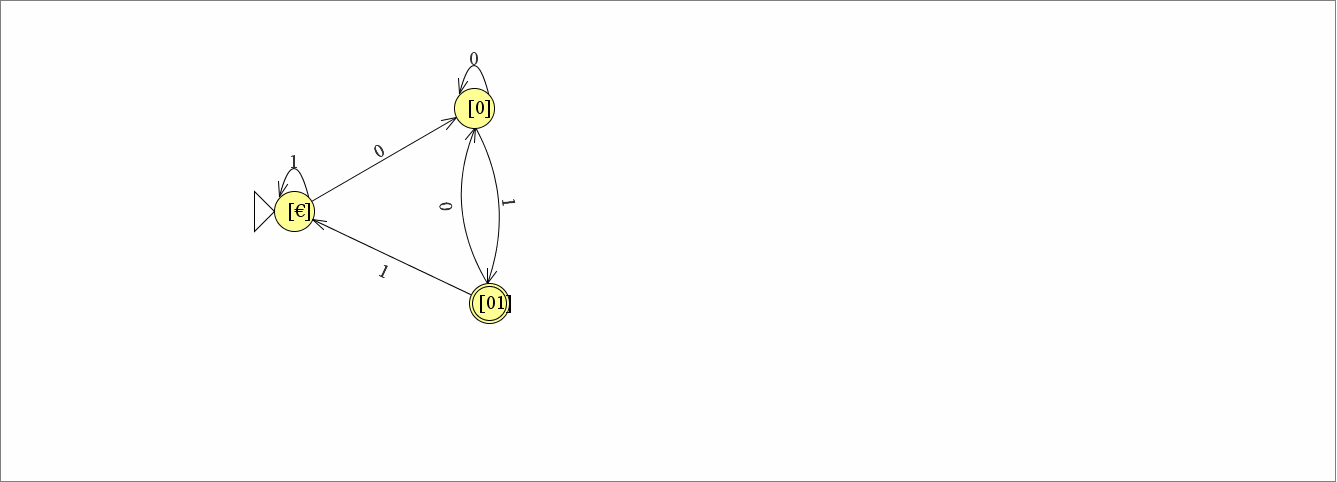
Como vemos, las clases de equivalencia, o estados del automata son **infinitos**, chocando con el concepto básico de autómata **finito** determinista. Así, llegamos a la conclusión de que este lenguaje es **no regular.**

**Ejemplo 2: L = { w | w termina en 01}, es decir, el lenguaje que posea palabras que terminen en 01. En Ʃ = {0,1}**

Al escribir algunas clases de equivalencia, nos damos cuenta de que sólo existen 3 posibles escenarios, ya que lo único que nos importa es en qué acaba la palabra.

* Todas aquellas palabras que no terminan en 0 ni en 01. Lo denotamos como [€]. **Ej: 1, 011, 100011, 11, etc.**
* Todas aquellas palabras que terminan en 0. Lo denotamos como [0]. **Ej: 0, 110, 010, 10, etc.**
* Todas aquellas palabras que terminan en 01. Lo denotamos como [01]. **Ej: 01, 0001, 0101101, etc.**

Afortunadamente, hemos acabado. El universo será Ʃ\* = [€] U [0] U [01]

Siendo estas finitas, es posible incluso construir la tabla de transiciones para un automata finito. Llegamos a la conclusión de que este lenguaje es **regular.**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| ->[€] | [0] | [€] |
| [0] | [0] | [01] |
| \*[01] | [0] | [€] |

**Myhill – Nerode vs Lema de bombeo**

Mientras que en el Teorema Myhill – Nerode vamos analizando y determinando todos los posibles casos en que puede caer un subconjunto de palabras de algun alfabeto, segun la condicion establecida por el lenguaje, en el Lema del bombeo simplemente seleccionamos una única vez una n, formamos la cadena resultante y la seccionamos en tres subcadenas: x, y, z, para así alterar la longitud de la subcadena intermedia y, y ver si la cadena que resulto sigue perteneciendo al lenguaje.

Un punto negativo es cuando se tienen dos valores (digamos n, m) que irán variando la longitud de una cadena, pues el proceso de vuelve largo, confuso y a veces se llega a un resultado incorrecto.

En Myhill – Nerode no es necesaria esta manipulación de cadenas, pues su funcionamiento tiende un poco más a la teoría de conjuntos. Otra ventaja que tiene este teorema es que casi de forma automatica, nos da el automata (en su forma más mínima posible) correspondiente al lenguaje, siempre y cuando este sea regular, mientras que el de bombeo no.

**Conclusiones**

De cierta manera, tanto el Teorema Myhill – Nerode como el Lema de Bombeo tienen sus ventajes y desventajas. Por ejemplo, cuando se trata de algun lenguaje que posee gran cantidad de clases de equivalencia, pero al final resulta uno regular, Myhill – Nerode se puede volver un poco largo y tedioso.

Sin embargo, al tratarse de muy pocos casos, como en el ejemplo de arriba, resulta muy cómo y eficaz, pues con el mismo procedimiento logramos determinar: regularidad de un lenguaje y su automata finito (tabla de transicion y diagrama).

Personalmente, siento la necesidad de emplear antes que éste el lema de Bombeo, debido a que, desde mi punto de vista, posee un mayor grado de precisión al analizar un lenguaje. Aunque todo depende de que tan grande sea la palabra a la que vamos a ir cambiando o “bombeando” su longitud.

Por otro lado, a veces es posible adivinar la regularidad de un lenguaje con tan solo verlo, siendo que, si nos damos cuenta de inmediato que las clases de equivalencia de éste serán infinitas, podemos optar por utilizar el teorema y rapidamente justificar su irregularidad.

**2.- Forma Normal de Greibach (FNG)**

Uno de los mayores retos que surgen cuando se esta trabajando con Gramáticas Independientes del Contexto (GIC), es su implementación. Esto como consecuencia de redundancias o mala optimización.

Para esto existen dos formas de normalización de Gramáticas, que nos ayudaran a evitar problemas posteriores: la Forma Normal de Chomsky y la Forma Normal de Greibach.

De manera formal, una GIC esta en FNG si:

* La variable inicial no es recursiva
* La gramatica no tiene variables inútiles
* La gramatica no tiene producciones Ɛ (excepto S -> Ɛ)
* Todas las producciones son de la forma: A -> a ó A -> aB1B2B3...Bk

Donde Bi son no terminales en ƩN, y a es terminal en ƩT.

Esto quiere decir, que todas las reglas de produccion deben iniciar con un símbolo terminal, sin recursividad y sin cadenas vacías de longitud cero. Una gramatica regular izquierda no cumple con esta forma, mientras que una gramatica regular puede o no cumplir esta forma.

Cabe mencionar que para una GIC G que no esta en FNG, siempre existirá otra GIC G’ en FNG que será equivalente.

Vamos a construir un algoritmo que nos permitirá realizar lo anterior, nos apoyaremos de la siguiente gramatica:

1. Si no lo esta, debemos establecer nuestro GIC G a su definicion formal: G = {ƩT, ƩN, S, P}
2. Eliminamos reglas S -> Ɛ
3. Eliminamos la recursividad a izquierdas en G.
4. Ordenamos todos los símbolos no terminales ΣN: A1< A2<….< An
5. Clasificamos las reglas de P en:

* Aquellas que ya empiezan con un símbolo terminal.
* Aquellas donde Ai -> Ajx, donde i < j y x es cualquier símbolo.
* Aquellas donde Ai -> Ajx, donde i > j y x es cualquier símbolo.

1. Seleccionamos las reglas del tercer grupo; donde la posicion Ai sea mínima en el ordenamiento de los símbolos no terminales, sustituimos Aj.
2. Si existen reglas recursivas a la izquierda, las eliminamos agregando un nuevo símbolo no terminal a ƩT, y colocandolo al principio de la ordenacion ΣN. Repetir hasta que no existen reglas recursivas a la izquierda.
3. Sustituimos Aj en las reglas del segundo grupo.
4. Si aparece algún símbolo terminal c en el cuerpo de las reglas del primer grupo (sin contar primera posicion), se crea una nueva variable no terminal B, formada por la regla B -> c
5. Se sustituye c en todos los cuerpos de las reglas de P (excepto primera posicion)
6. Se vuelve añadir la regla S -> Ɛ si existe.

**Ejemplo:** Hallar la GIC en FNG equivalente de la sig. Gramatica:

A->Ba B->CA | a C->AB | b

**1, 2 y 3. Esta bien definida y no posee reglas recursivas a la izquierda.**

**4.- Ordenamos A<B<C**

**5.- Grupo 1:** C->b, B->a. **Grupo 2:** A->Ba, B->CA. **Grupo 3:** C->AB.

**6.- Se eliminan reglas del grupo 3:**

C -> AB -> BaB -> CAaB | aaB

**7.- Se eliminan las reglas recursivas a izquierdas**

C -> CAaB | aaB | b -> aaBD | bD | aaB | b

D -> AaBD | AaB

**Se añade D en la primera posición en el orden: D<A<B<C**

**8.- Se eliminan las reglas del grupo 2 desde mayor a menor**

C -> CAaB | aaB | b -> aaBD | bD | aaB | b

B -> CA -> aaBDA | bDA | aaBA | bA

A -> Ba -> aaBDAa | bDAa | aaBAa | bAa | aa

D -> AaBD -> aaBDAaaBD | bDAaaBD | aaBAaaBD | bAaaBD | aaaBD

D -> AaB -> aaBDAaaB | bDAaaB | aaBAaaB | bAaaB | aaaB

**9 y 10.- Se sustituyen símbolos terminales**

A -> aEBDAE | bDAE | aEBAE | bAE | aE

B -> aEBDA | bDA | aEBA | bA | a

C -> aEBD | bD | aEB | b

D -> aEBDAEEBD | bDAEEBD | aEBAEEBD | bAEEBD | aEEBD | aEBDAEEB | bDAEEB | aEBAEEB | bAEEB | aEEB

E -> a

**11.- G no tiene S -> Ɛ.**

**Por lo tanto, G esta normalizada en FNG.**

**FNG vs Forma Normal de Chomsky**

La FNG nos es útil para la construccion del automata de pila relacionado a una GIC, mientras que con la FNC los árboles de derivacion del GIC serán árboles binarios, que nos facilitara la implementacion del GIC en un analizador sintactico.

Ahora bien, mientras que en la premisa principal del FNG es que cada regla de produccion debe comenzar con un símbolo terminal y que no existan éstos en el cuerpo de cada regla de produccion, en el FNC esta prohibido tener más de dos varibales no terminales en cada regla de produccion.

Eso sí, en ambas se busca eliminar la recursividad obsoleta que pudiese provocar procesos ineficientes o repetidos, y que la GIC este lo más optimizada posible.

**Conclusiones**

Las formas normalizadas de Chomsky y de Greibach nos son útiles porque gracias a ellas, podemos tener una mejor comprensión de una Gramática, desde un punto de vista computacional, ya que éstas nos ayudarán en la construccion de automatas y determinación de lenguajes regulares equivalentes.