La Salle Universidad Bolivia

TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

MSc. Ing. Martha Fernández Montaño

INTRODUCCIÓN A LA TEORIA DE AUTÓMATAS Y LENGUAJES FORMALES

MSc. Ing. Martha Fernández Montaño
CIA, CISA, CGAP, CGEIT, CISM

INTRODUCCIÓN A LA TEORIA DE AUTÓMATAS Y LENGUAJES FORMALES

1º Edición
Martha Fernández Montaño
Año 2017

Editorial: Universidad La Salle Bolivia
Calle Jorge Carrasco, esquina Las Palmas, No. 450- Zona Bolognia
Teléfono: (591) – 2 – 2723588

Depósito Legal: 4-1-3943-16
ISBN: 978-99974-63-09-8
Impresión: Universidad La Salle Bolivia
La Paz- Bolivia

A mi hija Michelle, que es el tesoro más grande que Dios me ha concedido.

Y a mis padres, Jorge y Litzia, que han sido una luz y sabia guía en mi vida.

AGRADECIMIENTOS

Agradezco a Dios por la oportunidad que me da de dar a conocer esta parte de las matemáticas y la teoría informática, la Teoría de Autómatas, y permitirme deleitarme con el descubrimiento de leyes perfectas como todas las que Él ha establecido en el universo y que me dejan asombrada por la comprensión de que todo lo que podamos aprender, es solo una pequeña parte de la infinita sabiduría de Dios.

Agradezco a mi hija, Michelle, por todo su cariño y apoyo, así como su inspiración y ayuda en la finalización de este libro.

Agradezco con mucho cariño y muy especialmente a mis padres, Jorge y Litzia, por estar siempre a mi lado y por su apoyo incondicional.

Agradezco a la Lic. Martha Heredia, por el tiempo dedicado a la lectura de este libro y sus comentarios profesionales y didácticos.

Agradezco al Ing. Roberto Mamani por lograr un diseño tan adecuado para la portada de este libro.

Agradecimientos especiales a la Universidad La Salle por hacer posible la publicación de este libro, especialmente al Lic. Bruno Vargas por animarme a publicar y a la Lic. Wilma Peñafiel por ocuparse de todos los detalles de publicación.

PREFACIO

La misión de toda institución educativa universitaria es encarar procesos de formación profesional con el fin de servir a la sociedad y brindar investigaciones que contribuyan al desarrollo en nuestra sociedad.

La universidad La Salle con el fin de dar cumplimiento, apoya a los estudiantes y docentes en esta línea, y la publicación de este libro es una muestra de este trabajo.

La teoría de autómatas y los lenguajes formales están dentro del campo científico del campo de la Informática Teórica, y es un campo que no ha sufrido cambios en las últimas décadas como suele pasar con otras áreas de la Informática, pero es la base para muchas áreas como la computabilidad, autómatas celulares, redes neuronales, inteligencia artificial, etc.

Este libro muestra los conceptos más importantes en el estudio de esta rama, de una manera sencilla, completa y práctica, ya que cada concepto viene acompañado de un ejemplo demostrativo y una propuesta, que deja al lector desarrollar su capacidad.

La aplicabilidad está en cada uno de nosotros, apoyado de la investigación, se puede presentar proyectos basados en esta teoría que ayuden dar soluciones a las problemáticas actuales de nuestra vida cotidiana.

Felicitamos a la autora del libro por todos sus aportes que son fruto de una trayectoria profesional de más de 25 años en el ejercicio de la docencia en diferentes instituciones educativas y las investigaciones realizadas y plasmadas en este libro, que será de mucha utilidad para la comunidad docente-estudiantil.

Lic. Martha Heredia

CONTENIDO

PRÓLOGO

"Las matemáticas son el alfabeto con el cual Dios ha escrito el Universo".

- Galileo Galilei (1564 – 1642 A.D.)

Dos importantes aspectos de la vida que hacen que valga la pena vivirla: La oportunidad de dejar volar la imaginación para crear algo nuevo. Y la capacidad de descubrir y admirar el Universo que nos rodea.

Existe algo indispensable tanto para crear como para entender el mundo natural ya existente. Aunque considerada por algunos como ciencia para las *mentes cerradas y cuadradas*, ¿no es, más bien, una mente cerrada la que no es capaz de ver **las matemáticas**, describiendo el mundo natural, y permitiendo la creación de todo lo que hace nuestros días más felices? Siempre involucrada en todo, sea uno capaz de verlo o no. En todas las artes y las ciencias, (desde la música hasta la astronomía). En el mundo y fuera de él (desde el núcleo de una célula hasta la galaxia más lejana). Todo encaja, todo tiene sentido, pues son *lógicas*. Y al estudiarlas, uno se encuentra rozando la *perfección*.

El estudio de la teoría de autómatas abre todo un nuevo mundo hacia las matemáticas, abstrayéndolas de las máquinas físicas para analizar diferentes soluciones a algún problema. Después, se puede construir algo

ya más *palpable,* que nos permite *ver* las computaciones. Sí, requiere de una comprensión más profunda de las matemáticas, es verdad, y es justamente por esta razón que muchos no utilizan esta herramienta tan útil. Pero si a uno le pudieran enseñar el tema de manera *sencilla*, si uno tan solo pudiera comprenderlo sin mucho esfuerzo, no cabrían dudas en cuanto a qué tan fascinante es el tema y que es conocimiento que vale la pena compartir.

Este libro, busca justamente eso. Busca compartir conocimiento de un tema matemático más profundo, pero eliminando toda redundancia. Busca llegar a los demás, pero sin agregarles complicaciones innecesarias. Busca que sea un tema entendido y aprendido, sin necesidad de lágrimas ni sudor.

MSc. Ing. Martha Fernández Montaño
CIA, CISA, CGAP, CGEIT, CISM

CAPITULO 1

FUNDAMENTOS MATEMÁTICOS

1.1 Teoría de Conjuntos

Un conjunto es una colección de elementos, no necesariamente con alguna relación. El símbolo $\in$ es utilizado para indicar pertenencia, es decir, si tenemos $x \in X$ indica que x es elemento o elemento del conjunto X. Una diagonal atravesando el símbolo indica que x no es elemento de X.: $x \notin X$. Dos conjuntos son iguales si ellos contienen los mismos elementos. Un conjunto puede ser definido explícita o implícitamente:

X = { 1, 2, 3 } (explícitamente)

Y = { n : n = m^2 donde m es un número natural } (implícitamente,

Números naturales ($\mathbb{N}$) = {1, 2, 3, …)

1.1.1 Propiedades de conjuntos

El ***conjunto vacío*** es denotado por $\varnothing$, es el conjunto que no tiene elementos y puede ser definido explícitamente por $\varnothing$ = { }. Un conjunto está determinado por los elementos que contiene y el orden en el cual son representados es irrelevante. La repetición de elementos de un conjunto dentro del mismo no afecta al conjunto.

Un conjunto Y es un ***subconjunto*** de X, escrito $Y \subseteq X$, si cualquier elemento de Y es también elemento de X. El conjunto vacío es subconjunto de cualquier conjunto. Cualquier conjunto X es subconjunto de sí mismo. Si Y es un subconjunto de X y $Y \neq X$, entonces Y es llamado un subconjunto propio de X.

El conjunto de todos los subconjuntos de X es llamado el ***conjunto potencia*** de X y es denotado por 2^X.

Sea X = {1, 2, 3}, el conjunto potencia de X es:

$2^X = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{3,1\},\{1,2,3\}\}$

La ***igualdad*** de conjuntos puede ser definida como sigue: los conjuntos X y Y son iguales si $X \subseteq Y$ y $Y \subseteq X$. Lo que establece que cualquier elemento de X es también elemento de Y y viceversa.

1.1.2 Operaciones de Conjuntos

Existen operaciones entre conjuntos que nos permiten la construcción de nuevos conjuntos:

La ***unión*** de dos conjuntos está definida por:

$$X \cup Y = \{ z : z \in X \text{ o } z \in Y \}.$$

Como el operador o es inclusivo, esto indica que cuando z es elemento de $X \cup Y$ es porque z es elemento de X o de Y o de ambos.

La ***intersección*** de dos conjuntos genera el conjunto que tiene los elementos comunes a ambos conjuntos. Se define como sigue:

$$X \cap Y = \{ z : z \in X \text{ y } z \in Y \}.$$

Cuando la intersección de dos conjuntos es vacía, se dice que los conjuntos son ***disjuntos***.

La ***diferencia*** de los conjuntos X y Y, $X - Y$, consiste de los elementos de X que no están en Y.

$$X - Y = \{ z : z \in X \text{ y } z \notin Y \}.$$

Sea X un subconjunto de U. El ***complemento*** de X con respecto a U es el conjunto de elementos de U que no están en X. El complemento de un conjunto es denotado por X^c.

Las leyes de ***DeMorgan*** denotan las relaciones entre unión, intersección y complemento, cuando X y Y son subconjuntos de un conjunto U y su complemento es tomado de U.

i. $(X \cup Y)^c = X^c \cap Y^c$

ii. $(X \cap Y)^c = X^c \cup Y^c$

Ejercicio

Sea $X = \{1, 2, 3\}$ y $Y = \{2, 3, 4, 5\}$. X^c y Y^c denotan el complemento de X y Y con respecto a $\mathbb{N}$ (conjunto de números naturales). Encuentre la unión, intersección, diferencia entre ambos, complementos, intersección de los complementos y el complemento de la unión.

1.2 Producto Cartesiano, Relaciones y Funciones

Las relaciones y funciones matemáticas se definen como subconjuntos del producto cartesiano de conjuntos. Por tanto, comenzaremos por estudiar la operación sobre los conjuntos.

1.2.1 Producto Cartesiano

El ***producto cartesiano*** es una operación entre conjuntos que construye un conjunto que consiste de pares ordenados de elementos de los dos conjuntos. El producto cartesiano de los conjuntos X y Y, denotado $X \times Y$, es definido como:

$$X \times Y = \{(x, y) : x \in X \text{ y } y \in Y\}$$

1.2.2 Relaciones

Una ***relación binaria*** entre X y Y es un subconjunto de $X \times Y$. Por ejemplo, el orden de los números naturales puede ser usado para generar una relación LE (menor que) sobre el conjunto $\mathbb{N} \times \mathbb{N}$. Esta relación es el subconjunto de $\mathbb{N} \times \mathbb{N}$ definido por:

$$LE = \{ (i, j) : i, \leq \ j \text{ e } i, \ j \in \mathbb{N} \}$$

La notación $(i, j) \in LE$ indica que i es menor que j. (1,5), (2,7) $\in$ LE.

El producto cartesiano puede ser generalizado para construir conjuntos nuevos a partir de cualquier número finito de conjuntos. Si $x_1, x_2, ..., x_n$ son ***n*** elementos, entonces $(x_1, x_2, ..., x_n)$ es llamada tupla ordenada de orden n. Un par ordenado es llamado tupla ordenada de orden 2. El producto cartesiano de n conjuntos $X_1, X_2, ..., X_n$ es definida como:

$$X_1 \times X_2 \times ... \times X_n = \{(x_1, x_2, ..., x_n) : x_i \in X_i, \text{ para } i = 1, 2, \ldots, n\}$$

Una relación de orden n sobre $X_1, X_2, ..., X_n$ es un subconjunto de $X_1 \times X_2 \times ... \times X_n$.

Ejercicio

Sea $X = \{1, 2, 3\}$ y $Y = \{a, b\}$, encontrar $X \times Y$, $Y \times X$, $Y \times Y$ y $X \times Y \times Y$.

1.2.3 Funciones

Una ***función*** de un conjunto X a un conjunto Y es un mapeo de elementos de X a elementos de Y. Una función f de X a Y es denotada por $f : X \rightarrow Y$. El elemento de Y asignado por la función f a un elemento $x \in X$ es denotado por $f(x)$. Se utiliza la notación $f(x) = y$ para indicar que y es el valor asignado a x por la función f.

El conjunto X es llamado el ***dominio*** de la función. El ***rango*** de f es el subconjunto de Y que contiene los elementos de Y que son asignados a los elementos de X. Así, el rango de una función $f : X \rightarrow Y$ es el conjunto $\{ y \in Y : y = f(x)$ para alguna $x \in X \}$.

Podemos generalizar la definición para más conjuntos. El dominio de una función puede estar formado por el producto cartesiano de dos o más conjuntos:

$f : X_1 \times X_2 \times ... \times X_n \rightarrow Y$.

Cuando esto ocurre tenemos una función de n-variables. El valor de la función con variables $x_1, x_2, ..., x_n$ es denotada $f(x_1, x_2, ..., x_n)$. Las variables $x_1, x_2, ..., x_n$ son llamadas los argumentos de la función. Una función f relaciona elementos de su dominio con elementos del rango de f.

1.2.3.1 Propiedades de las funciones

Una ***función*** f de X a Y es una relación binaria sobre $X \times Y$ que satisface las siguientes propiedades:

i) Para cada $x \in X$ hay una $y \in Y$ tal que $(x, y) \in f$.

ii) Si $(x, y_1) \in f$ y $(x, y_2) \in f$, entonces $y_1 = y_2$.

Se utiliza la notación $f(x) = y$ para indicar que y es el valor asignado a x por la función f.

Ejercicio

Sea $X = \{1, 2\}$ y $Y = \{a, b, c\}$, encuentre funciones que se puedan definir entre los dos conjuntos.

Una función $f : X \rightarrow Y$ es ***suryectiva*** si para cada elemento de X existe un elemento distinto en el rango, es decir, f es suryectiva si $x_1 \neq x_2$ implica que $f(x_1) \neq f(x_2)$.

Una función $f : X \rightarrow Y$ es ***inyectiva*** si el rango de f incluye todos los elementos del conjunto Y.

Una función $f : X \rightarrow Y$ es ***biyectiva*** si es suryectiva e inyectiva.

Por ejemplo, considere las siguientes funciones que están definidas de $\mathbb{N}$ a $\mathbb{N}$ - {1}.

- $f(n) = 2n + 1$ Es suryectiva, ya que el rango consiste de los números impares, pero no es inyectiva, ya que los números pares no pertenecen al rango.
- $g(n) = 2$ si $n = 1$

  $g(n) = n$ si $n \neq 0$

  $g(n)$ es inyectiva ya que todos los naturales positivos se encuentran en el rango, pero no es suryectiva, ya que $n = 1$ y $n = 2$ tienen la misma imagen.
- $s(n) = n + 1$ es biyectiva, ya que es suryectiva e inyectiva

1.3 Conjuntos Contables y No-contables

La ***cardinalidad*** es la característica que nos da el tamaño de un conjunto, es decir, la cardinalidad de un conjunto es el número de elementos en el conjunto. Se puede demostrar que dos conjuntos tienen el mismo número de elementos estableciendo una función biyectiva entre los elementos de los conjuntos, o sea definiendo una biyección.

$a \rightarrow 1$

$b \rightarrow 2$

$c \rightarrow 3$

lo que demuestra que los conjuntos $\{a, b, c\}$ y $\{1, 2, 3\}$ tienen el mismo tamaño. Esta forma de comparar conjuntos funciona para conjuntos con un número finito o infinito de elementos.

i. Dos conjuntos X y Y tienen la misma cardinalidad si existe una función biyectiva que mapea X a Y. *Card* (X) = *card* (Y). Entonces decimos que X y Y son equinumerosos.
ii. Si Y tiene un subconjunto Z tal que card (X) = card (Z). Si card (X) < card (Y) y $X \neq Y$, entonces X tiene una cardinalidad menor que Y.

Para establecer la cardinalidad de conjuntos finitos e infinitos usaremos ***secuencias*** de elementos de un conjunto:

Sea $J = \{1, 2, 3, \ldots\}$ el conjunto de los números naturales o enteros positivos y $J_n = \{1, 2, \ldots, n\}$, el conjunto formado por los n primeros enteros positivos., donde $J_0 = \varnothing$; una secuencia sobre el conjunto X es una función f: $J \rightarrow X$,

donde una secuencia es representada por $f(1)$, $f(2)$, $f(3)$, …, $f(i)$, …, donde normalmente se utiliza la notación $x_1, x_2, ..., x_{i, ...}$, para $x_i \in X$. Una *secuencia finita de longitud n* sobre el conjunto X es una función $f: J_n \rightarrow X$, usualmente escrita como $x_1, x_2, ..., x_n$, para $x_i \in X$. La *secuencia de longitud cero* es la función $f: \varnothing \rightarrow X$.

Si tenemos dos conjuntos finitos X y Y, entonces $\text{card}(X) < \text{card}(Y)$ si y solo si $\text{card}(X) = \text{card}(J_m)$ y $\text{card}(Y) = \text{card}(J_n)$, donde $m < n$, podemos representar $\text{card}(X)$ y $\text{card}(Y)$ por los enteros m y n.

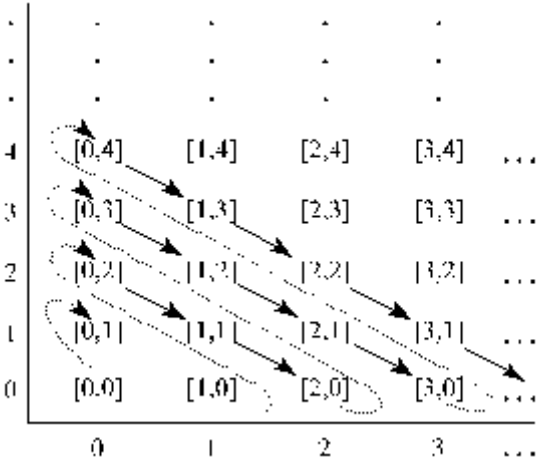
Un conjunto X es ***finito*** si $\text{card}(X) = \text{card}(J_n)$ para $n \geq 0$, en tal caso decimos que X tiene una cardinalidad n.

Un conjunto es ***infinito*** si no es finito. Un conjunto X es ***enumerable*** o ***infinitamente contable*** si se puede definir una función biyectiva con los números naturales, esto es si $\text{card}(X) = \text{card}(J)$. Un conjunto es ***contable*** si es finito o enumerable. Un conjunto X es enumerable si sus elementos pueden ser acomodados en una secuencia infinita $x_1, x_2, ..., x_i, ...$, para $x_i \in X$, en la cual cada elemento de X aparece solo una vez. Un conjunto es ***incontable*** si no es contable

Ejercicios

1. Pruebe que el conjunto de números naturales impares es enumerable o infinitamente contable con respecto a $\mathbb{N}$.

2. Pruebe que el conjunto de números enteros es enumerable con respecto a $\mathbb{N}$. Si 1 → 1, 2 → -1, 3 → 2, 4 → -2 ….



Los puntos de un arreglo bidimensional infinito puede ser usado para mostrar que $\mathbb{N} \times \mathbb{N}$, el conjunto de pares ordenados de números naturales es enumerable. El arreglo es construido colocando en las accisas los números naturales (incluyendo el 0), donde la posición del i-ésimo elemento en la accisa horizontal y el j-ésimo elemento en la accisa vertical

representan el par ordenado (i, j). En algunos casos, aunque 0 no es un número natural formalmente, es conveniente incluirlo en el conjunto.

Los elementos del arreglo pueden ser listados secuencialmente siguiendo las flechas en el diagrama, creando la siguiente correspondencia:

0	1	2	3	4	5	6	7	…
↓	↓	↓	↓	↓	↓	↓	↓	
(0,0)	(0,1)	(1,0)	(0,2)	(1,1)	(2,0)	(0,3)	(1,2)	…

que demuestra la contabilidad de $\mathbb{N} \times \mathbb{N}$. Se define la función biyectiva que convierte el par ordenado (i, j) al número natural $((i + j)(i + j + 1) / 2) + i$.

Los conjuntos de interés en la teoría de lenguajes y computabilidad son aquellos que son infinitos o enumerables.

i. La unión de dos conjuntos contables es contable.
ii. La unión de un número infinito contable de conjuntos contables es contable.
iii. El producto Cartesiano de dos conjuntos contables es contable.
iv. El conjunto de subconjuntos finitos de un conjunto contable es contable.
v. El conjunto de secuencias de longitud finita formadas de elementos de un conjunto contable no vacío es infinitamente contable.

1.4 Definiciones Recursivas

Muchos de los conjuntos involucrados en la generación de lenguajes contienen un número infinito de elementos, por lo que se debe definir un conjunto infinito de tal forma que permita que los elementos del conjunto puedan ser construidos y manipulados. Para lograrlo, se debe tener un método de definición que permita a una computadora, que carece de intuición, generar y entender las propiedades de los conjuntos.

Una ***definición recursiva*** de un conjunto X especifica un método para construir los elementos de un conjunto. La definición utiliza dos componentes: La base y un conjunto de operaciones, donde la base consiste de un conjunto finito de elementos que son explícitamente designados como elementos de X y las operaciones son usadas para construir nuevos elementos del conjunto a partir de los elementos previamente definidos. La recursividad que define al conjunto X consiste de todos los elementos que pueden ser generados a partir de los elementos básicos por la aplicación de un número finito de veces de las operaciones.

Una definición recursiva de $\mathbb{N}$, el conjunto de números naturales, es construído usando la función sucesor s.

i. Paso base: $1 \in \mathbb{N}$.

ii. Paso recursivo: si $n \in \mathbb{N}$, entonces $s(n) = n + 1 \in \mathbb{N}$.

1 es un número natural por la definición del paso base. Un nuevo número natural está definido en términos de un número definido previamente. El paso recursivo garantiza que el conjunto contiene solo aquellos elementos que pueden ser obtenidos a partir de 1 usando el operador sucesor. La esencia de un procedimiento recursivo es definir procesos o estructuras complicados en términos de instancias más simples del mismo proceso o estructura.

1.5 Inducción Matemática

Para establecer las relaciones entre los elementos de conjuntos y operaciones entre conjuntos se requiere construir pruebas que verifiquen las propiedades establecidas como hipótesis. Es imposible probar que una propiedad se mantiene para cualquier elemento en un conjunto infinito al considerar cada elemento en forma individual. El principio de inducción matemática da suficientes condiciones para probar que una propiedad prevalece para cualquier elemento en un conjunto definido recursivamente. La inducción utiliza la familia de conjuntos anidados generados por el proceso recursivo para extender una propiedad desde la base al conjunto completo.

1.5.1 Principio de inducción matemática

Sea A un conjunto, con las siguientes propiedades, definidas por recursión, usando la función sucesor s.

i. Paso base: $1 \in A$.
ii. Paso recursivo: si $n \in A$, entonces $s(n) = n + 1 \in A$.

1 es un elemento de A por la definición del paso base y cada nuevo elemento de A, definido en términos de un número definido previamente, es un número natural. Por lo tanto podemos concluir que el conjunto A es en realidad el conjunto $\mathbb{N}$. Si comparamos esta definición de A con la definición anterior de $\mathbb{N}$, podemos comprobar que definen el mismo conjunto.

1.5.2 Demostración por inducción matemática

Sea A un conjunto definido por recursividad y sea **P** una propiedad definida para los elementos de A. Si puede ser mostrado que:

i. **P** se cumple para el elemento base.
ii. **P** se cumple para cada elemento generado a partir del elemento base, hasta un cierto elemento
iii. **P** también se cumple para cualquier elemento generado a partir del último del paso anterior,

entonces, por el principio de inducción matemática, **P** se cumple para cualquier elemento en A, o sea, para cualquier elemento en $\mathbb{N}$.

Una prueba inductiva consta de tres pasos:

a. ***Paso base:*** Probar que la propiedad **P** se cumple para cada elemento del conjunto base. Esto corresponde a establecer condiciones indicadas en (i) en la definición del principio de la inducción matemática, con $n = 1$ o $n = 0$ (si se considera al 0 como elemento de $\mathbb{N}$.

b. ***Hipótesis de inducción:*** Declaración de la hipótesis inductiva. La hipótesis inductiva es el asumir que la propiedad **P** se cumple para cualquier elemento generado a partir del elemento base, con $n = k$.

c. ***Paso de inducción:*** El paso inductivo prueba, usando la hipótesis inductiva, que **P** puede ser extendida al elemento posterior del paso anterior, o se $n = k + 1$. Completando el paso inductivo se satisfacen los requerimientos del principio de inducción matemática.

Por ejemplo, usamos el método de demostración matemática para probar que:

$$0 + 1 + \ldots. + n = n(n + 1)/ 2.....$$

Usando la notación de sumatoria podemos escribir la expresión como sigue:

$$\sum_{i=0}^{n} i = n(n + 1) / 2$$

Paso base: La base es $n = 0$. La relación es explícitamente establecida por el cálculo del valor de cada uno de los lados de la igualdad:

$$\sum_{i=0}^{0} i = 0 = 0(0 + 1) / 2$$

Hipótesis inductiva: Asumimos que para los valores $n = 1, 2, \ldots k$

$$\sum_{i=0}^{k} i = k(k + 1) / 2$$

Paso inductivo: Necesitamos probar que

$$\sum_{i=0}^{k+1} i = (k + 1)((k + 1) + 1) / 2 = (k + 1)(k + 2) / 2$$

La hipótesis inductiva establece el resultado para la suma de la secuencia conteniendo n o menos enteros. Combinando la hipótesis inductiva con las propiedades de la suma obtenemos:

$$\sum_{i=0}^{k+1} i = \sum_{i=0}^{k} i + (k + 1) \text{ (asociatividad)}$$

$$= k (k + 1) / 2 + (k + 1) \text{ (hipótesis inductiva)}$$

$$= (k + 1)(k / 2 + 1) \text{ (propiedad distributiva)}$$

$$= (k + 1)(k + 2) / 2$$

Debido a que las condiciones del principio de inducción matemática han sido establecidas, concluimos que el resultado se cumple para todos los números naturales. Cada paso en la prueba debe seguir las propiedades previamente establecidas por los operadores o la hipótesis inductiva. La

estrategia de una hipótesis inductiva es manipular la fórmula para contener una instancia de la propiedad aplicada al caso más simple.

Otro ejemplo, para demostrar por inducción matemática que $n! > 2^n$, para $n \geq 4$:

Paso base: $n = 4$ entonces $4! = 24 > 16 = 2^4$

Hipótesis Inductiva: Asumimos que $n! > 2^k$ para todos los valores $n = 4, 5, \ldots, k$.

Paso Inductivo: Necesitamos probar que $(k+1)! > 2^{k+1}$.

$(k + 1)! = k! (k + 1)$

$> 2^k (k + 1)$ (hipótesis inductiva)

$> 2^k(2)$ (ya que $k + 1 > 2$)

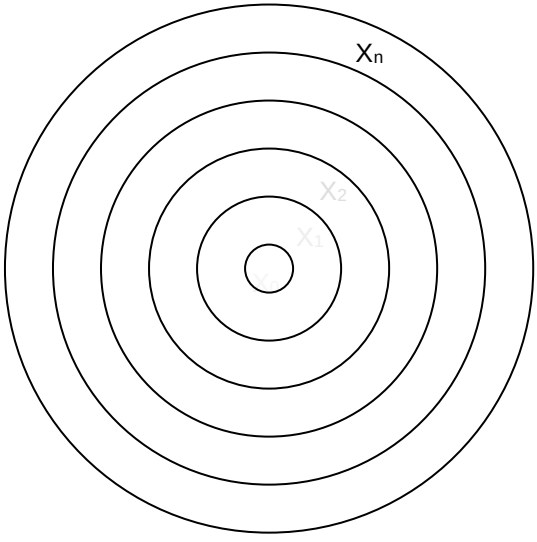
$= 2^{k+1}$

Podemos utilizar el definiciones inductivas para definir los elementos de un conjunto, resultado de alguna operación o función, generado por medio de la recursividad.

Por ejemplo, podemos dar una definición por inducción matemática para la suma de m y n, la cual se establece como sigue:

i. Paso base: Si $n = 1$, entonces $m + n = m + 1$.
ii. Paso inductivo: $m + s(n) = s(m + n)$.

Secuencia anidada de conjuntos en una definición recursiva.



Supongamos que deseamos obtener la suma de 3 + 2, donde estos números, aplicando la operación sucesor, son obtenidos como $s(s(1))$ y $s(1)$, respectivamente, la suma puede ser calculada recursivamente como:

$s\,(s(1)) + s(1)$

$= (s(s(1)) + 1)$

$= (s(s(s(1)))$

$= 5$

$X_0 = \{ x : x \text{ es el elemento base } \}$

$X_{i+1} = \{ x : x \text{ puede ser generado por } i + 1 \text{ operaciones } \}$

$X = \{ x : x \in X_j \text{ para alguna } j \geq 0 \}$

El operador sucesor puede ser usado recursivamente para definir por inducción relaciones sobre el conjunto $\mathbb{N} \times \mathbb{N}$. Para mostrarlo, utilizaremos la relación LE, donde buscaremos los pares ordenados (i, j) en los cuales $i < j$.

La relación LE está definida por :

i. Paso base: $(1,2) \in LE$.
ii. Paso inductivo: Si $(n, m) \in LE$, entonces $(n, s(m)) \in LE$ y $(s(n), s(m)) \in LE$.

Si utilizamos la unión indicada en la recursión, para generar los elementos de LE, obtenemos la siguiente secuencia de LE_i:

$LE_0 = \{ (1, 2) \}$

$LE_1 = LE_0 \cup \{(1, 3), (2, 3)\}$

$LE_2 = LE_1 \cup \{(1, 4), (2, 4), (3, 4)\}$

$LE_3 = LE_2 \cup \{(1, 5), (2, 5), (3, 5), (4, 5)\}$

…

$LE_i = LE_{i-1} \cup \{(j, i + 1) : j = 0, 1, \ldots, i\}$

…

La generación de un elemento en un conjunto definido mediante recursividad puede no ser única, por ejemplo, el par ordenado $(2, 4) \in LE_2$ puede ser generado a través de dos secuencias de operaciones distintas.

Base:	(1, 2)	(1,2)
1:	(1, s(2)) = (1, 3)	(s(1), s(2)) = (2, 3)
2:	(1, s(3)) = (1, 4) (2, s(3)) = (2, 4)	(s(1), s(3)) = (2, 4) (s(2), s(3)) = (3; 4)

Ahora usaremos una definición por inducción matemática para aquellos pares ordenados de $\mathbb{N} \times \mathbb{N}$, donde su segundo componente sea 3, 4, 5 ó 6.

i. Paso base: (1, 3), (1, 4), (1, 5), (1, 6) están en X.

ii. Paso inductivo: Si $(n, m) \in X$, entonces $(s(n), m) \in X$.

$X_i = \{ (j, 3), (j, 4), (j, 5), (j, 6) : j = 0, 1, \ldots. i \}$

Ejercicios

1. Dé una definición por inducción matemática de la relación de igualdad sobre $\mathbb{N} \times \mathbb{N}$ usando el operador sucesor.
2. Dé una definición por inducción matemática de la relación mayor que sobre $\mathbb{N} \times \mathbb{N}$ usando el operador sucesor.
3. Dé una definición por inducción matemática del conjunto de puntos (m, n) que están en la línea $n = 3m$ en $\mathbb{N} \times \mathbb{N}$ usando el operador sucesor.
4. Sea X un conjunto finito, dé una definición por inducción matemática del conjunto de subconjuntos de X. Use la unión como el operador en la definición.
5. Pruebe por inducción matemática que:

   $2 + 5 + 8 + \ldots + (3n + 1) = n(3n + 1)/2$ para $n > 0$

Ejercicios del capítulo

1. Sea C un conjunto de conjuntos definido de la siguiente manera:
   - $\varnothing \in C$
   - Si $S_1 \in C$ y $S_2 \in C$, entonces $\{S_1, S_2\} \in C$
   - Si $S_1 \in C$ y $S_2 \in C$, entonces $S_1 \times S_2 \in C$
   - Nada más pertenece a C.

a) Explique si $\{\varnothing, \{\varnothing\}\} \in C$

b) Dé el ejemplo de dos conjuntos S tal que $S \in C$ y $|S| > 1$

c) Explique si C contiene algún conjunto infinito

d) Explique si C es contable o incontable

2. Sea $A = \{a, b, c, d\}$, encuentre:
   a) 2^A
   b) Π_A
   c) El número de elementos de un conjunto potencia, en general
   d) El número de particiones de un conjunto, en general

3. Demuestre por inducción matemática que toda relación de orden parcial en un conjunto finito no vacío tiene por lo menos un elemento mínimo.

4. Probar por inducción matemática que $1 + 2^n < 3^n$ para $n > 2$

5. Sea $P = \{A, B\}$ el conjunto de dos variables booleanas. El conjunto E está formado por expresiones booleanas bien formadas a partir de conjunciones y disyunciones (or y and) sobre P, cuya definición por inducción matemática es como sigue:

   i. Paso base: $A, B \in E$.

   ii. Paso inductivo: Si $u, v \in E$, entonces $(u \vee v) \in E$ y $(u \wedge v) \in E$.

   iv.

   a) Encuentre las expresiones booleanas que se consiguen con $n = 0, 1$ y 2.

   b) Probar por inducción matemática que para cualquier expresión booleana en E, el número de ocurrencias de variables es mayor en uno al número de operadores. Para una expresión u, $n_v(u)$ es el número de variables en u y $n_o(u)$ es el número de operadores en u.

   c) Probar por inducción matemática que para cualquier expresión booleana en E, el número de paréntesis izquierdos es igual al número de paréntesis derechos.

CAPITULO 2

ALFABETOS Y LENGUAJES

2.1 Alfabetos y Cadenas

Una ***cadena*** sobre un conjunto X es una secuencia finita de símbolos tomados de X. Las cadenas son los objetos fundamentales en la definición de lenguajes.

El conjunto de elementos a partir del cual son construidas las cadenas es llamado el ***alfabeto*** del lenguaje, donde un alfabeto está formado por un

conjunto finito de objetos indivisibles y es denotado por Σ.

Sea Σ un alfabeto, Σ^*, el conjunto de cadenas generadas a partir de Σ, es definido como sigue:

i. Paso base: $\lambda \in \Sigma^*$.

ii. Paso inductivo: Si $w \in \Sigma^*$ y $a \in \Sigma$, entonces $wa \in \Sigma^*$.

La longitud de una cadena w es el número de elementos en la cadena o el número de veces que se aplicó el paso recursivo, la cual es denotada como *longitud* (w) o como $|w|$. Si Σ consiste de n elementos, entonces hay n^k cadenas de longitud k en Σ^*.

Ejercicio

Sea Σ = { a, b, c }. Encuentre cadenas generadas en Σ^*.

2.1.1 Operaciones de Cadenas

Sea $u,v \in \Sigma^*$. La ***concatenación*** de u y v, escrita uv, es una operación binaria sobre Σ^* definida como sigue:

i. Paso base: Si *longitud* (v) = 0, entonces $v = \lambda$ y $uv = u$.

ii. Paso inductivo: Sea v una cadena con *longitud* (v) = $n + 1 > 0$. Entonces $v = wa$, para alguna cadena w con longitud n y $a \in \Sigma$ y $uv = (uw)a$.

Sean $u, v, w \in \Sigma$. Entonces $(uv)w = u(vw)$. Propiedad de *asociatividad*.

Ejercicio

Sean $u = ab$, $v = ca$ y $w = bb$. Encuentre uv, vw, $(uv)w$, $u(vw)$.

Las subcadenas pueden ser definidas utilizando la operación de concatenación. Intuitivamente, u es una subcadena de v si u aparece o forma parte de v. Formalmente, u es una ***subcadena*** de v si hay cadenas x y y tal que $v = x\,u\,y$. Un ***prefijo*** de v es una subcadena u en la cual x es la cadena nula en la descomposición de v, esto es $v = u\,y$. Similarmente, u es un ***sufijo*** de v si $v = x\,u$.

Sea u una cadena en Σ^*. La ***cadena reversa*** de u, denotado por u^R, es definido como sigue:

i. Paso base: *longitud* $(u) = 0$, Entonces $u = \lambda$ y $\lambda^R = \lambda$.

ii. Paso inductivo: Si *longitud* $(u) = n + 1 > 0$, entonces $u = wa$, para alguna cadena w con longitud n y $a \in \Sigma$ y $u^R = aw^R$.

Ejercicios

1. Encuentre el reverso de $abbc$.
2. Sea $u, v \in \Sigma^*$. Demuestre por inducción matemática que $(uv)^R = v^R u^R$.

Sea u una cadena en Σ^*. La ***cadena repetida*** de u, denotada por u^i, es definida como sigue:

i. Paso base: $i = 0$, Entonces $u^i = \lambda$.

ii. Paso inductivo: Si $i = n + 1 > 0$, entonces $u^i = u^n u$.

Ejercicios

Sean $u = ab$, $v = ca$ y $w = bb$. Encuentre u^0, u^1 y u^2.

2.2 Lenguajes

Un ***lenguaje*** normalmente está formado por un subconjunto del conjunto de todas las posibles cadenas que se pueden generar a partir del alfabeto. Un lenguaje generado a partir de un alfabeto Σ es un subconjunto de Σ^*.

2.3 Especificación Finita de Lenguajes

Hemos definido un lenguaje como un conjunto de cadenas generadas a partir de un alfabeto. Los lenguajes de interés son aquellos que sus cadenas tienen una forma específica, donde las formas aceptables definen la sintaxis del lenguaje. La especificación de un lenguaje requiere de una descripción no ambigua de las cadenas del mismo. Un lenguaje finito puede ser descrito solamente con enumerar sus elementos, pero algunos

lenguajes infinitos son definidos estableciendo requerimientos sintácticos y pueden ser definidos inductivamente.

Por ejemplo, el lenguaje L de cadenas formadas a partir de {a, b}, en el cual cada cadena comienza con a y tiene longitud par, es definido como:

i. Paso base: aa, $ab \in L$.

ii. Paso inductivo: Si $u \in L$, entonces uaa, uab, uba, $ubb \in L$.

Otro ejemplo, como el lenguaje L formado por cadenas sobre {a, b}, en el cual cada ocurrencia de b está precedida por una a. Por tanto, λ, a, $abaab$ están en L y bb, bab, abb no están. Podemos definirlo como sigue:

i. Paso base: $\lambda \in L$.

ii. Paso inductivo: Si $u \in L$, entonces ua, $uab \in L$.

2.3.1 Operaciones de Lenguajes

La ***unión*** de lenguajes X y Y, denotada por $X \cup Y$, es el lenguaje

$$X \cup Y = \{u : u \in X \text{ o } u \in Y\}$$

La ***intersección*** de lenguajes X y Y, denotada por $X \cap Y$, es el lenguaje

$$X \cap Y = \{u : u \in X \text{ y } u \in Y\}$$

La ***diferencia*** de lenguajes X y Y, denotada por $X - Y$, es el lenguaje

$$X - Y = \{u : u \in X \text{ y } u \notin Y\}$$

La ***complementación*** de X, denotada por $\overline{X}$, es el lenguaje

$$\overline{X} = \{u : u \in \Sigma^* \text{ y } u \notin X\}$$

La ***concatenación*** de lenguajes X y Y, denotada por XY, es el lenguaje

$$XY = \{uv : u \in X \text{ y } v \in Y\}$$

La ***cerradura o estrella de Kleene*** de X, denotada por X^*, es el lenguaje

$$X^* = \{u_1u_2u_3u_4.....u_n : u_i \in X \text{ para } i = 0, 1, 2, 3, 4, ..., n\}$$

La ***cerradura transitiva*** de X, denotada por X^+, es el lenguaje

$$X^+ = \{u_1u_2u_3u_4.....u_n : u_i \in X \text{ para } i = 1, 2, 3, 4, ..., n\}$$

Ejercicio

Sea $X = \{a, b, c\}$ y $Y = \{abb, ba\}$, encontrar $X \cup Y$, $X \cap Y$, $X - Y$, $Y - X$, $\overline{X}$, $\overline{Y}$, X^* y Y^*.

2.4 Expresiones Regulares

Sea Σ un alfabeto, una ***expresión regular*** sobre Σ está definida como sigue:

i. $\varnothing$ es una expresión regular sobre Σ.
ii. λ es una expresión regular sobre Σ.

iii. a, para cualquier $a \in \Sigma$, es una expresión regular sobre Σ.

iv. Si u y v son expresiones regulares sobre Σ, . Entonces $(u \cup v)$ también es una expresión regular.

v. Si u y v son expresiones regulares sobre Σ, . Entonces $(u \cup v)$ también es una expresión regular sobre Σ.

vi. Si u y v son expresiones regulares sobre Σ, . Entonces (uv) también es una expresión regular sobre Σ.

vii. Si u es una expresión regular sobre Σ, . Entonces (u^*) también es una expresión regular sobre Σ.

viii. Nada más es una expresión regular.

Debido a que la unión y la concatenación son asociativas, los paréntesis pueden ser omitidos.

Según la definición anterior, el lenguaje representado por una expresión regular, denotado por $L(u)$ puede definirse como:

i. $L(\varnothing) = \varnothing$.

ii. $L(\lambda) = \lambda$.

iii. $L(a) = a$.

iv. $L(\mathrm{X} \cup \mathrm{Y}) = L(\mathrm{X}) \cup L(\mathrm{Y})$.

v. $L(\mathrm{XY}) = L(\mathrm{X})L(\mathrm{Y})$.

vi. $L(\mathrm{X}^*) = (L(\mathrm{X}))^*$.

Sea Σ un alfabeto, la ***clase de lenguajes regulares*** $\Re$ sobre Σ, está definida como sigue:

i. $\varnothing \in \Re$.

ii. $\{\lambda\} \in \Re$.

iii. $\{a\} \in \Re$, para cualquier $a \in \Sigma$.

iv. Si X y Y $\in \Re$, entonces $X \cup Y$, XY, $X^* \in \Re$.

v. Sea $\wp$ es un conjunto de lenguajes tal que $\varnothing \in \wp$, $\{\lambda\} \in \wp$, $\{a\} \in \wp$, para cualquier $a \in \Sigma$, y $\wp$ sea cerrado para la unión, concatenación y estrella de Kleene, entonces $\Re \subseteq \wp$.

Esto implica que un lenguaje es regular si y solo si puede ser representado por una expresión regular.

2.4.1 Identidades de las Expresiones Regulares

1. $\varnothing u = u\varnothing = \varnothing$
2. $\lambda u = u\lambda = u$
3. $\varnothing^* = \lambda$
4. $\lambda^* = \lambda$
5. $u \cup v = v \cup u$
6. $u \cup \varnothing = u$
7. $u \cup u = u$
8. $u^* = (u^*)^*$
9. $u(v \cup w) = uv \cup uw$

10. $(u \cup v)w = uw \cup vw$
11. $(uv)^*u = u(vu)^*$
12. $(u \cup v)^* = (u^* \cup v)^*$
    $= u^* (u \cup v)^* = (u \cup vu^*)^*$
    $= (u^*v^*)^* = u^*(vu^*)^*$
    $= (u^*v)^*u^*$

2.5 Aplicaciones

Las expresiones regulares tienen diversas aplicaciones, principalmente para representar las operaciones de un analizador léxico o de un editor de textos. Los objetos utilizados en los programas, pueden expresarse como expresiones regulares casi sin excepción.

Por ejemplo, los identificadores en ALGOL, podrían expresarse de la siguiente manera:[1]

(letra) (letra ∪ dígito)*

donde letra es el conjunto de letras mayúsculas y minúsculas del alfabeto y dígito es el conjunto de dígitos del 0 al 9.

[1] Hopcroft & Ullman, Introduction to Automata Theory, Languages and Computation, p. 45

Ejercicios del capítulo

1. Obtenga expresiones regulares que representen el conjunto descrito:
   a. El conjunto de cadenas sobre {a, b, c} en el cual las $a´s$ preceden a las $b´s$ y estas a su vez preceden a las $c´s$.
   b. El conjunto de cadenas de longitud 2 sobre {a, b} en el cual las $a´s$ preceden a las $b´s$.
   c. El conjunto de cadenas sobre {a, b, c} que no contienen la subcadena aa.
   d. El conjunto de cadenas sobre {a, b} que no comienzan con la subcadena aaa.
   e. El conjunto de cadenas de longitud impar sobre {a, b} que contienen la subcadena bb.

2. Defina el lenguaje que puede ser obtenido a partir de {a, b}, cuyas cadenas contienen la subcadena bb.

3. Defina el lenguaje formado por las cadenas que comienzan con aa o terminan con bb, a partir de {a, b}.

4. Dé una definición por inducción matemática del conjunto de cadenas generadas a partir de {a, b} que contienen igual número de $a´s$ y $b´s$.

5. Dé una definición por inducción matemática del conjunto $\{a^i b^j : 0 < i < j\}$

6. Sea L un lenguaje generado a partir de {*a*, *b*} y definido como sigue:

   i. Paso base: $\lambda \in L$.

   ii. Paso inductivo: Si $u \in L$ entonces $aaub \in L$.

   a. Obtenga los tres primeros elementos de L a partir del paso base.

   b. Dé una definición implícita del conjunto de cadenas generadas por la definición por inducción matemática.

7. Sea X = {*aa*, *bb*} y Y = {*l*, *b*, *ab*}.

   a. Obtenga las cadenas en el conjunto XY.

   b. Obtenga las cadenas del conjunto Y* de longitud menor o igual a 3.

   c. ¿Cuántas cadenas de longitud 6 hay en X*?

CAPITULO 3

AUTÓMATAS FINITOS

Un procedimiento que determina si una cadena de entrada está en un lenguaje es llamado un ***reconocedor de lenguaje***. Las propiedades comunes para todas las máquinas incluyen el procesamiento de una entrada y la generación de una salida. La entrada a este tipo de máquinas es una cadena formada a partir de un alfabeto y el resultado arrojado indica si la entrada es aceptada o no.

3.1 Máquinas de Estados Finitos

Para poder definir una máquina de estados finitos, utilizaremos el comportamiento de una máquina vendedora de café:

La entrada a la máquina consiste de monedas de 10 centavos, 20 centavos y 50 centavos, de tal forma que cuando son insertados 60 centavos, el botón dispensador puede ser activado para obtener una taza de café. Si el total de dinero insertado en la máquina excede 60 centavos, ésta no da cambio. Esta máquina carece de memoria tal y como es concebida en una computadora, pero ella sabe que son requeridos 10 centavos para poder abrir la tapa, si es que previamente se habían introducido 50 centavos. Este conocimiento es logrado por la alteración de los estados internos de la máquina cuando una entrada es recibida y procesada. Una máquina de estados representa el estado de un cálculo en marcha. La operación interna de la máquina vendedora puede ser descrita por las interacciones de los siguientes estados:

- *se necesitan 60 centavos*. El estado de la máquina antes de que cualquier moneda sea insertada.
- *se necesitan 50 centavos*. El estado al que se llega cuando se insertó una moneda de 10 centavos.
- *se necesitan 40 centavos*. El estado al que se llega después de haber insertado dos monedas de 10 centavos o una de 20 centavos.

- *se necesitan 20 centavos*. El estado después de cuatro monedas de 10 centavos o una de 20 y dos de 10 centavos o dos de 20 centavos.
- *se necesitan 10 centavos*. El estado después de una de 50 centavos, cinco de 10 centavos, dos de 20 y una de 10 centavos o una de 20 y tres de 10 centavos.
- *se necesitan 0 centavos*. El estado indica que al menos se han insertado 60 centavos.

Al insertar una moneda a la máquina ocasiona que ésta cambie su estado. Cuando 60 centavos o más han sido insertados, se llega al estado *se necesitan 0 centavos* y el botón dispensador puede ser activado. Dicho estado es llamado ***estado final*** o ***de aceptación***.

El diseño de la máquina debe representar cada uno de los componentes simbólicamente, de tal forma que en lugar de tener una secuencia de monedas, la entrada a la máquina abstracta es una cadena de símbolos. Un grafo dirigido etiquetado conocido como ***diagrama de estados***, puede ser utilizado para representar las transiciones de la máquina. Para esto, hacemos que los nodos de dicho diagrama sean los estados antes mencionados. El nodo "*se necesitan* m *centavos"* es representado por m. Los arcos son etiquetados como d, v o c representando la entrada de una moneda de 10, 20 ó 50 centavos, respectivamente. Un arco del nodo x al nodo y etiquetado con z indica el procesamiento de la entrada z cuando la máquina está en el estado x dando como resultado la transición al estado y.

La entrada a la máquina consiste de cadenas a partir de {*c, d, v* }*. La secuencia de estados alcanzados durante el procesamiento de una cadena de entrada puede ser trazada siguiendo los arcos en el diagrama de estados. La cadena *dcdc* es aceptada pero la cadena *nndn* no lo es.

3.2 Autómata Finito no Determinístico

Un ***autómata finito no determinístico*** es un quíntuple $M = (K, \Sigma, \Delta, s, F)$, donde:

K es un conjunto finito de ***estados***

Σ es un conjunto finito de símbolos llamado ***alfabeto***

$s \in K$ es un estado llamado ***estado inicial***

F es un subconjunto de K cuyos elementos son llamados ***estados finales*** *o* ***aceptantes***

Δ es una relación que es subconjunto del producto cartesiano $K \times \Sigma* \times K$ conocida como la ***relación de transición***.

Un AFND lee la entrada de izquierda a derecha, de tal forma que una vez que un símbolo de entrada ha sido procesado, no tiene efectos en operaciones posteriores. En cualquier punto de las operaciones, el resultado depende solo del estado actual y de la cadena que falta por leer. Esta combinación es llamada ***configuración de la máquina*** y es representada por el par ordenado (q_i, w), donde q_i es el estado actual y $w \in \Sigma*$ es la entrada no procesada. El símbolo |— es utilizado para indicar las

transiciones efectuadas por el AFND (***pasos de configuración***). El símbolo |—* es utilizado para indicar la ***cerradura reflexiva y transitiva del paso de configuración***.

Una cadena de entrada es aceptada si existen transiciones que procesan la cadena completa y la máquina termina en un estado de aceptación. Una cadena está dentro del lenguaje aceptado por la máquina no determinística si existen transiciones que la llevan del estado inicial a un estado final. El ***lenguaje*** de un AFND M, denotado por L(M), es el conjunto de cadenas aceptadas por M. Esto es:

$$L(M) = \{ w: \text{existe una transición } (s, w) \vdash^* (p, \lambda) \text{ con } p \in F\}.$$

Una cadena de entrada es rechazada si existen transiciones que procesan la cadena completa y la máquina termina en un estado de no aceptación, o si la cadena no puede procesarse completamente. Es decir, cuando:

- $(s, w) \vdash^* (p, \lambda)$ con $p \notin F$
- $(s, w) \vdash^* (p, u)$ con $p \in F$ y no se puede continuar procesando
- $(s, w) \vdash^* (p, u)$ con $p \notin F$ y no se puede continuar procesando

El diagrama de estados de un AFND $M = (K, \Sigma, \Delta, s, F)$ es un grafo dirigido etiquetado G definido por las siguientes condiciones:

i. Los nodos de G son elementos de K.
ii. Las etiquetas en los arcos de G son elementos de $\Sigma*$. Solo las transiciones necesarias son graficadas.
iii. s es el nodo inicial.
iv. F es el conjunto de nodos aceptantes.

v. Hay un arco del nodo q_i a q_j etiquetado con v si $(q_i, v, q_j) \in \Delta$.

Ejercicio

Considere el siguiente AFND.

$M = (K, \Sigma, \Delta, s\ F)$

$K = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

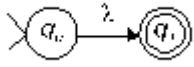
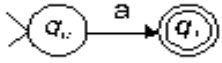
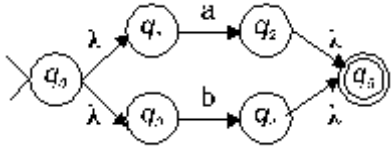
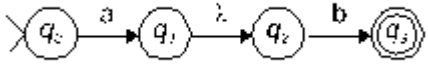
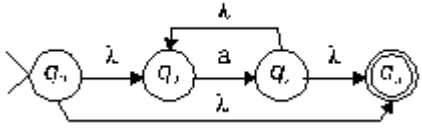
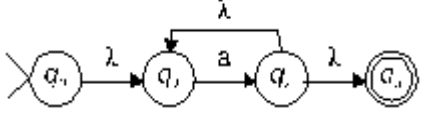
$F = \{q_2\}$

$\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, b, q_2),\}$

Encuentre las transiciones posibles para la cadena *ababb*.

3.2.1 Transiciones Lambda

Un AFND puede permitir transiciones sin requerir una entrada para que sea procesada, una transición de esta forma es llamada ***transición lambda***. La clase de máquinas no determinísticas que utilizan transiciones lambda es denotada AFND-λ. Un AFND con transiciones lambda es una quíntuple $M = (K, \Sigma, \Delta, s, F)$, donde K, Δ, s y F son lo mismo que un AFND. La relación de transición es una relación de $K \times (\Sigma* \cup \{\lambda\})$ x K. Este tipo de autómatas nos permiten convertir una expresión regular a un AFND-λ aplicando el método de Thompson, el cual se aplica de la siguiente manera:

Expresión Regular	**AFND- λ**
λ	
a	
A ∪ b	
ab	
a*	
a+	

3.3 Autómata Finito Determinístico

Un ***autómata finito determinístico*** (AFD) es un quíntuple $M = (K, \Sigma, \delta, s, F)$, donde:

K es un conjunto finito de ***estados***

Σ es un conjunto finito de símbolos llamado ***alfabeto***

$s \in K$ es un estado llamado ***estado inicial***

F es un subconjunto de K cuyos elementos son llamados ***estados finales*** o ***aceptantes***

δ es una función de $K \times \Sigma$ a K conocida como la ***función de transición***.

Un AFD lee la entrada de izquierda a derecha, de tal forma que una vez que un símbolo de entrada ha sido procesado, no tiene efectos en operaciones posteriores. En cualquier punto de las operaciones, el resultado depende solo del estado actual y de la cadena que falta por leer. Esta combinación es llamada ***configuración de la máquina*** y es representada por el par ordenado (q_i, w), donde q_i es el estado actual y $w \in \Sigma^*$ es la entrada no procesada. El símbolo |— es utilizado para indicar el ***paso de una configuración a otra*** en el AFND. El símbolo |—* es utilizado para la ***cerradura transitiva del paso de configuraciones*** en el AFND.

Una cadena leída es aceptada por un AFD si existen pasos de configuraciones que procesan la cadena completa y la máquina termina en

un estado de aceptación. Una cadena pertenece al lenguaje de la máquina determinística si existen transiciones que la llevan del estado inicial a un estado final. El ***lenguaje*** de un AFD M, denotado por $L(M)$, es el conjunto de cadenas en Σ^* aceptadas por M. Esto es:

$$L(M) = \{ w: \text{existe una transición } (s, w) \vdash^* (p, \lambda) \text{ con } p \in F\}.$$

Una cadena de entrada es rechazada si existen transiciones que procesan la cadena completa y la máquina termina en un estado de no aceptación. Es decir, cuando:

$(s, w) \vdash^* (p, \lambda)$ con $p \notin F$

El diagrama de estados de un AFD $M = (K, \Sigma, \delta, s, F)$ es un grafo dirigido etiquetado G definido por las siguientes condiciones:

i. Los nodos de G son elementos de K.
ii. Las etiquetas en los arcos de G son elementos de Σ.
iii. s es el nodo inicial.
iv. F es el conjunto de nodos aceptantes.
v. Hay un arco del nodo q_i a q_j etiquetado con a si $\delta(q_i, a) = q_j$.
vi. Para cualquier nodo q_i y símbolo $a \in \Sigma$, hay exactamente un arco etiquetado con a saliendo de q_i.

Dos máquinas que aceptan el mismo lenguaje se dicen que son ***equivalentes***.

Ejercicios

1. Considere el siguiente AFD.

M= (K, Σ, δ, *s*, F)

K = {q_0, q_1}

Σ = {*a*, *b*}

F = {q_1}

q	σ	δ(*q*,σ)
q_0	*A*	q_1
q_0	*B*	q_0
q_1	*A*	q_1
q_1	*B*	q_0

Encuentre las transiciones para la cadena *aba*.

2. Considere el siguiente AFD, donde L(M) = $(a \cup b)^*bb(a \cup b)^*$

M = (K, Σ, δ, *s*, F)

K = {q_0, q_1, q_2}

Σ = {*a*, *b*}

F = {q_2}

q	σ	*δ*(*q*, *σ*)
q_0	*a*	q_0
q_0	*b*	q_1
q_1	*a*	q_0
q_1	*b*	q_2
q_2	*a*	q_2
q_2	*b*	q_2

Encuentre las transiciones para las cadenas *abab* y *abba*.

3. Construya un diagrama de AFD para cada uno de los siguientes lenguajes:
   a. $(ab)^*ba$
   b. $aa(a \cup b)^+bb$
   c. $(l \cup \_)\ (l \cup d \cup \_)^*$
   d. El lenguaje formado por cadenas que tienen un número par de *a´s* con un alfabeto {*a*, *b*}
   e. El lenguaje formado por cadenas que comienzan con *aa* o terminan con *bb*.

3.4 Autómatas Finitos Equivalentes

Aunque las representaciones parezcan muy distintas en algunos casos, los lenguajes que pueden ser representados por los autómatas finitos determinísticos AFDs y los autómatas finitos no determinísticos AFND son exactamente los mismos.

Teorema: Para todo AFND existe un AFD equivalente.[2]

El poder obtener un AFND-λ a partir de una expresión regular, nos permite generar un AFD en base al AFND-λ, dado como resultado por la

[2] Lewis & Papadimitiou Elements of the Theory of Computation p. 60

metodología de Thompson. Para poder realizarlo, tenemos dos funciones: La ***cerradura lambda*** de un estado q_i, denotada $E(q_i)$, está definida por:

$E(qi) = \{qj : (qi, \lambda) \vdash\!\!\!-^* (qj, \lambda),\}$, o lo que es lo mismo:

$E(qi) = \{qj : (qi, w) \vdash\!\!\!-^* (qj, w),\}$

Y la ***función de transición por entrada t*** de un AFND-λ es una función de $2^K \times \Sigma$ a 2^K, donde $Q \in 2^K$ definida por:

$$d(Q_i, a) = \bigcup E(\delta(q_i, a)\ ), \text{ donde } q_i \in Q_i$$

Los pasos para encontrar el autómata finito equivalente, utilizando estas funciones son los siguientes:

1. Primero, eliminamos las cadenas que pudieran existir en el AFND-λ, $M = (K, \Sigma, \Delta, s, F)$ y obtenemos un autómata equivalente $M' = (K', \Sigma, \Delta', s', F')$ sin cadenas en las transiciones.
2. Encontramos el estado inicial del AFD equivalente $M'' = (K'', \Sigma, \delta, s'', F'')$, mediante $s'' = E(s')$.
3. Encontramos el resto de las transiciones del AFD mediante la función de transición por entrada t, donde, comenzando con el estado inicial encontrado s'', mediante $\delta(Qi, a) = \bigcup E(qj) : qj \in Qi$ y $(qi, a, qj) \in \Delta'$
4. Encontramos los estados finales del AFD, que estarán dados por todos los estados cuya intersección con los estados finales de M' no sea vacía, es decir: $Qi \cap F' \neq \varnothing$.

Ejercicio

Construya un AFD a partir de AFND-λ obtenidos con anterioridad.

CAPITULO 4

LENGUAJES LIBRES DE CONTEXTO

4.1 Jerarquía de Chomsky

Chomsky clasificó las gramáticas y lenguajes dentro de cuatro familias jerárquicamente ordenadas como modelos potenciales del lenguaje natural:

Gramáticas sin restricciones	tipo 0
Gramáticas sensitivas al contexto	tipo 1
Gramáticas libres de contexto	tipo 2
Gramáticas regulares	tipo 3

Las restricciones colocadas en las reglas aumentan con el tipo al que pertenece la gramática. Cada clase de lenguaje en la jerarquía de Chomsky es acompañada por el lenguaje generado por una familia de gramáticas y por un tipo de máquina que lo acepta.

Gramáticas	**Lenguajes**	**Máquinas**
Gramáticas Tipo 0, gramáticas estructura-frase gramáticas sin restricciones	Lenguajes recursivamente enumerables	Máquinas de Turing, Máquinas de Turing no deterministas
Gramáticas Tipo 1, gramáticas sensitivas al contexto gramáticas monótonas	Lenguajes sensitivos al contexto	Autómata lineal restringido
Gramáticas Tipo 2, gramáticas libres de contexto	Lenguajes libres de contexto	Autómata De pila

Gramáticas Tipo 3, gramáticas regulares gramáticas lineales hacia la izquierda gramáticas lineales hacia la derecha	Lenguajes Regulares	Autómata finito determinístico, Autómata finito no determinístico

Una ***Gramática sin restricciones*** es un cuádruple de la forma (V, **Σ**, R, S), donde:

V es un ***alfabeto*** que contiene símbolos terminales y no terminales

Σ es un ***alfabeto*** que solo contiene símbolos terminales

R es un conjunto finito de ***reglas***

S es un elemento de V llamado el ***símbolo inicial***

Una producción de una gramática sin restricciones tiene la forma $u \rightarrow v$, donde $u \in (\mathbf{V} \cup \mathbf{\Sigma})^+$ y $v \in (\mathbf{V} \cup \mathbf{\Sigma})^*$. Los conjuntos **V** y **Σ** son disjuntos. Una gramática estructura-frase G = (**V**, **Σ**, R, **S**) es llamada **sensitiva al contexto** si cada producción tiene la forma $u \rightarrow v$, donde $u \in (\mathbf{V} \cup \mathbf{\Sigma})^+$ y $v \in (\mathbf{V} \cup \mathbf{\Sigma})^+$ y *longitud* (*u*) <= *longitud*(*v*). Una regla que satisface esta condición es llamada **monótona**.

4.2 Gramáticas y Lenguajes Libres de Contexto

Gramática es el conjunto de reglas que denotan un lenguaje, por lo que las gramáticas son utilizadas para generar cadenas formadas correctamente sobre un alfabeto preestablecido. El ***símbolo de inicio*** inicia el proceso de generar cadenas aceptables. Una ***Gramática Libre de Contexto*** es un cuádruple de la forma G = (V, **Σ**, R, S), donde:

V es un ***alfabeto*** que contiene símbolos terminales y no terminales

Σ es un ***alfabeto*** que solo contiene símbolos terminales

R es un conjunto finito de ***reglas***

S es un elemento de V llamado el ***símbolo inicial***

Σ $\subset$ V

V - Σ es el alfabeto que solo contiene símbolos no terminales

Una ***regla*** es un elemento del conjunto (V - Σ) x (V)*. La regla (A, *w*) es escrita A $\rightarrow$ *w*. Dado que la cadena vacía está incluida en (V)*, λ puede estar al lado derecho de la regla, tal que una regla de la forma A $\rightarrow$ λ es llamada ***regla nula*** o ***regla lambda***. ***Símbolos terminales*** son aquellos que en su descripción denotan lo que representan. ***Símbolos no terminales*** son variables cuyo valor está definido por una regla dentro de una gramática y deben ser reemplazados por otros símbolos para generar una cadena. Las convenciones para el uso de símbolos en las reglas son las siguientes:

- Los símbolos terminales en una gramática libre de contexto son representados por letras minúsculas.

- Los símbolos no terminales son representados por letras mayúsculas.

Ejercicio

Escriba algunas cadenas que puedan ser generadas por los siguientes conjuntos de reglas de producción.

S → *a*S*dd*	C → λ
S → A	S → *a*S
A → *b*A*c*	S → *b*A
A → *bc*	A → *a*A
C → *a*C	A → *b*C
C → *b*C	

4.2.1 Derivaciones

La manera en que se realiza la generación de cadenas a través de la aplicación de reglas es llamada **derivación**. Si tenemos una cadena de la forma uAv y una regla de la forma $A \rightarrow w$, entonces la aplicación de la regla en dicha cadena genera o deriva la cadena de la forma uwv, lo cual es denotado como $uAv \Rightarrow uwv$, donde el prefijo u y el sufijo v definen el contexto en el cual el no terminal A ocurre y $\Rightarrow$ es la función derivación.

Se dice que una gramática es libre de contexto cuando no existe restricción alguna con respecto a la aplicación de las reglas, es decir, una regla de *A* puede ser aplicada en cualquier parte de la gramática donde aparezca el símbolo no terminal *A* de (V - Σ), por lo que el contexto no establece limitación alguna sobre la aplicación de reglas. Una cadena *w* es derivada a partir de *v*, si podemos realizar derivaciones en una secuencia finita de reglas que transforma *v* a *w*, lo cual es llamado una ***computación***:

$$v \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = w$$

que, usando la cerradura reflexiva y transitiva de la función derivación, puede también representarse como:

$$v \Rightarrow w^*$$

4.2.2 Lenguajes Libres de Contexto

Un lenguaje es un conjunto de cadenas formadas a partir de un alfabeto. Un lenguaje generado por la gramática G es el conjunto de cadenas terminales derivadas a partir del símbolo inicial.

Sea G = (V, Σ, R, S) una gramática libre de contexto.

i. Una cadena $w \in (V)^*$ es una ***forma sentencial*** de G si se puede conseguir una derivación $S \Rightarrow^* w$ a partir de las reglas en G.

ii. Una cadena $w \in \Sigma^*$ es una ***sentencia*** de G si se puede conseguir una derivación $S \Rightarrow^* w$ a partir de las reglas en G.

iii. El ***lenguaje*** de G, denotado como L(G), es el conjunto de cadenas $\{w \in \Sigma^*: S \Rightarrow^* w\}$.

Las formas sentenciales son las cadenas derivables a partir del símbolo inicial de la gramática. Las sentencias son formas sentenciales que contienen sólo símbolos terminales. El lenguaje de una gramática es el conjunto de sentencias generadas por la gramática. Un conjunto de cadenas generadas a partir de un alfabeto Σ, es llamado ***lenguaje libre de contexto*** si hay una gramática libre de contexto que lo genera. Dos gramáticas son equivalentes si generan el mismo lenguaje.

La recursividad es necesaria en un conjunto finito de reglas para generar lenguajes infinitos y cadenas de longitud arbitraria y podemos notar que:

- Una regla de la forma $A \rightarrow uAv$ es llamada ***directamente recursiva***.
- Si existe una recursividad sobre A, entonces debe existir una regla A no recursiva para indicar el fin de ésta.
- Un símbolo no terminal A es llamado recursivo si se puede lograr una derivación de la forma $A \Rightarrow^* uAv$
- Una derivación de la forma $A \Rightarrow^* w \Rightarrow^* uAv$, donde A no es una subcadena de w, se denomina como ***indirectamente recursiva***.

Ejercicio

Sea G = (V, Σ, R, S), donde:

V = {S, A, *a*, *b*}

Σ = {*a*, *b*}

R = {S → AA,

A → AAA,

A → *b*A,

A → A*b*,

A → *a*}

encuentre todas las posibles derivaciones de la cadena *ababaa* en G.

4.2.3 Árbol de Derivación o Árbol Parse

Cuando al aplicar reglas en una derivación se realiza la sustitución del primer no terminal de izquierda a derecha, la derivación es llamada ***derivación más hacia la izquierda***. En caso de que la sustitución se realice derecha a izquierda, se llama ***derivación más hacia la derecha***. La característica esencial de una derivación no es el orden en el cual las reglas son aplicadas, sino la manera en la cual cada símbolo no terminal es descompuesto. La descomposición puede ser representada gráficamente por una derivación o un árbol parse.

Sea G = (V, Σ, R, S) una gramática libre de contexto y $S \Rightarrow^* w$, una derivación, el ***árbol de derivación***, AD, de $S \Rightarrow^* w$ es un árbol ordenado que puede ser construido iterativamente como sigue:

i. Inicializar AD con la raíz *S*.

ii. Si S → $u_1u_2u_3...u_n$ con $u_i \in V$ es una regla en G, entonces agregar $u_1u_2u_3...u_n$ como el hijo de S en el árbol.

iii. Si A → $u_1u_2u_3...u_n$ con $u_i \in V$ es una regla de G que puede aplicarse en la derivación a la cadena xAv, entonces agregar $u_1u_2u_3...u_n$ como el hijo de A en el árbol.

iv. Si A → λ es una regla de G que puede aplicarse en la derivación de la cadena xAv, entonces agregar λ como el hijo de A en el árbol.

Ejercicios

1. Diseñe los árboles de derivación para las generaciones de la cadena *ababaa* del ejercicio anterior. Distinga entre árboles con derivaciones más hacia la izquierda y más hacia la derecha. Un árbol de derivación puede ser usado para producir varias derivaciones que generan la misma cadena.

2. Sea G = (V, Σ, R, S), donde:

   V = {S, A, *a*, *b*}

$\Sigma = \{a, b\}$

$R = \{S \rightarrow aSa,$

$S \rightarrow aAa,$

$A \rightarrow bA,$

$A \rightarrow b\}$

encuentre $L(G)$

3. Si $L(G) = \{a^n b^m c^m d^{2n}: n \geq 0, m > 0\}$, encuentre las reglas de la gramática que generan dicho lenguaje.

4. Construya una gramática para reconocer la palabras palíndromas que se pueden obtener sobre $\{a, b\}$.

5. Sea $G = (V, \Sigma, R, S)$, donde:

$V = \{S, a, b\}$

$\Sigma = \{a, b\}$

$R = \{S \rightarrow aSb,$

$S \rightarrow aSbb,$

$S \rightarrow \lambda\}$

encuentre $L(G)$.

6. Sea $G = (V, \Sigma, R, S)$, donde:

$V = \{S, B, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow abSbcB,$

S → λ,

B → *b*B,

B → *b*}

encuentre L(G).

7. Sea G = (V, Σ, R, S), donde:

V = {S, A, *a*, *b*, *c*, *d*}

Σ = {*a*, *b*, *c*, *d*}

R = {S → *ab*S*c*,

S → A,

A → *c*A*d*,

A → *cd*}

a) Escriba la derivación de *ababccddcc*.

b) Construya el árbol de derivación para la misma cadena.

c) Defina L(G) usando notación de conjuntos.

8. Sea G = (V, Σ, R, S), donde:

V = {S, A, B, *a*, *b*}

Σ = {*a*, *b*}

R = {S → ASB,

S → λ,

A → *a*A*b*,

A → λ,

B → *b*B*a*,

B → *ba*}

a) Dé una derivación más hacia la izquierda de *aabbba*.

b) Dé una derivación más hacia la derecha de *abaabbbabbaa*.

c) Construya el árbol de derivación para las derivaciones de las dos cadenas anteriores.

d) Defina L(G) usando notación de conjuntos.

4.3 Transformación de Gramáticas

Se pueden realizar varias transformaciones en las gramáticas, para implementarlas más fácilmente. En cada transformación de G se debe conseguir una gramática G' equivalente.

4.3.1 Eliminación de Recursividad del Símbolo Inicial

La primera transformación que se aplica sobre una gramática, se establece sobre el símbolo inicial. Una regla recursiva de la forma S → *u*S*v*, permite que el símbolo inicial aparezca en varios pasos de la derivación. Lo que buscamos es encontrar G', tal que no existan derivaciones recursivas sobre S.

Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto. Hay una gramática $G' = (V', \Sigma, R', S')$ que satisface

i. $L(G) = L(G')$

ii. Las reglas de R' son de la forma:

$$A \rightarrow w.$$

donde $A \in (V' - \Sigma)$ y $w \in (V' - \{S'\})^*$.

Entonces tenemos que:

a. Si el símbolo inicial S no aparece en el lado derecho de una regla de G, entonces $G' = G$.

b. Si S es una variable recursiva, la recursividad del símbolo inicial debe ser eliminada, lo cual se realiza agregando una regla de la forma $S' \rightarrow S$, creando un nuevo símbolo inicial S', de modo que $G' = (V \cup \{S'\}, \Sigma, R \cup \{S' \rightarrow S\}, S')$. Las dos gramáticas generan el mismo lenguaje, ya que si u, una cadena cualquiera, derivable en G por una derivación $S \Rightarrow^* u$ puede ser obtenida por la derivación $S' \Rightarrow^* Su$.

Ejercicios

Para las siguientes gramáticas, encuentre una gramática G', donde no exista recursividad sobre el símbolo inicial.

1. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, A, B, C, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow aS,$

$S \rightarrow AB,$

S → AC,

A → *a*A,

A → λ,

B → *b*B,

B → *b*S,

C → *c*C,

C → λ}

2. G = (V, Σ, R, S), donde:

V = {S, A, B, C, *a*, *b*, *c*}

Σ = {*a*, *b*, *c*}

R = {S → ABC,

S → λ,

A → *a*A,

A → *a*,

B → *b*B,

B → A,

C → *c*C,

C → λ}

3. G = (V, Σ, R, S), donde:

V = {S, A, B, *a*, *b*}

Σ = {*a*, *b*}

R = {S → BSA,

S → A,

A → *a*A,

A → λ,

B → B*ba*,

B → λ}

4. Para la gramática siguiente, obtenga la derivación más hacia la izquierda de *aaaa*.

   G = (V, Σ, R, S), donde:

   V = {S, A, B, *a*, *b*}

   Σ = {*a*, *b*}

   R = {S → BSA,

   S → *a*B,

   B → *b*B,

   B → λ}

   Encuentre el lenguaje generado por la gramática y expréselo utilizando una expresión regular.

5. Para la siguiente gramática obtenga la derivación más hacia la izquierda de la cadena *aaaa*.

   G = (V, Σ, R, S), donde:

   V = {S, A, B, *a*, *b*}

   Σ = {*a*, *b*}

   R = {S → S*a*B,

   S → S*a*

S → *a*B

S → *a*

B → *b*B

B → *b*

Encuentre el lenguaje generado por la gramática y expréselo utilizando una expresión regular.

4.3.2 Eliminación de Reglas Lambda

Una variable que puede derivar la cadena nula es llamada ***variable anulable***. La longitud de una forma sentencial puede ser reducida por la aplicación de una regla lambda. Una gramática sin variables anulables es llamada ***gramática no contractable***, ya que la aplicación de una regla no puede reducir la longitud de la forma sentencial. Para obtener una gramática sin reglas lambda, primero tenemos que encontrar las variables anulables, para ello utilizamos el siguiente algoritmo:

Construcción del Conjunto de Variables Anulables

Entrada: Gramática libre de contexto G = (V, Σ, R, S)

VNUL = {A : A → λ ∈ R}

repetir

a. BUFF = VNUL

b. para cada variable A ∈ (V - Σ):

si hay una regla de la forma A → *w* y *w*

∈ BUFF* entonces

VNUL = VNUL ∪ {A}

hasta que VNUL = BUFF;

Ejercicio

Encuentre las variables anulables de las siguientes gramáticas:

1. G = (V, Σ, R, S), donde:

   V = {S, A, B, C, *a*, *b*, *c*}

   Σ = {*a*, *b*, *c*}

   R = {S → ACA,

   A → *a*A*a*,

   A → B,

   A → C,

   B → *b*B,

   B → *b*,

   C → *c*C,

   C → λ}

2. G = (V, Σ, R, S), donde:

   V = {S, A *a*, *b*}

   Σ = {*a*, *b*}

   R = {S → S*a*A,

   S → *a*A,

A → *b*A,

A → λ}

3. G = (V, Σ, R, S), donde:

V = {S, A *a*, *b*}

Σ = {*a*, *b*}

R = {S → *a*S,

S → AC,

S → AB,

A → *a*A,

A → λ,

B → *c*B,

B → λ,

C → *b*C,

C → *b*S}

Una gramática con reglas lambda es contractable. Para construir una gramática no contractable se debe agregar reglas para generar las cadenas cuyas derivaciones en la gramática original requieran la aplicación de reglas lambda.

En una regla, puede aparecer una variable A anulable en la derecha, como en:

B → *a*AB,

A → λ,

A → *b*C

En la derivación de cadenas, al usar la regla B, para la variable A puede utilizarse la regla lambda o la otra regla A, que es una cadena y por lo tanto no reduce la longitud de la cadena derivada. Anticipándose al uso de la regla lambda, podemos añadir la regla:

B → *a*B

a la gramática, y podemos anular la regla lambda. Las reglas equivalentes resultantes son:

B → *a*AB,

B → *a*B ,

A → *b*C

Si en las reglas, aparecen dos variables anulables a la derecha, se necesitan tres reglas adicionales para poder eliminar la regla lambda. Por ejemplo, tenemos las siguientes reglas en una gramática:

B → *a*AB*b*A,

A → λ,

A → *b*C

Al utilizar la regla B, una variable A, ambas variables A, o ninguna de ellas puede después derivarse con la regla lambda. Teniendo en cuenta todos los casos posibles, debemos añadir las siguientes reglas para poder eliminar la regla lambda:

B → *a*B*b*A,

B → *a*AB*b*,

B → *a*B*b*

Las reglas equivalentes resultantes son:

$B \rightarrow aABbA$,

$A \rightarrow bC$,

$B \rightarrow aBbA$,

$B \rightarrow aABb$,

$B \rightarrow aBb$

Los pasos antes mencionados nos permiten eliminar reglas lambda de una gramática G y nos dan como resultado una gramática G' donde se cumple que $L(G) = L(G')$, y por lo tanto, las dos gramáticas son equivalentes.

La única regla lambda que no puede eliminarse de una gramática es $S \rightarrow \lambda$, ya que esta regla está presente siempre que λ sea una cadena del lenguaje, es decir $\lambda \in L(G)$, y no existe otra forma de generar la cadena λ a partir de una gramática. Si una gramática contiene la regla $S \rightarrow \lambda$, pero no tiene ninguna otra regla lambda, es llamada una ***gramática esencialmente no contractable***.

La gramática obtenida a partir de G por la remoción de reglas lambda es denotada como G_L. Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto, entonces existe un algoritmo para construir una gramática libre de contexto equivalente $G_L = (V_L, \Sigma_L, R_L, S_L)$ que satisface:

i. $L(G_L) = L(G)$.

ii. S_L es una variable no recursiva.

iii. $A \rightarrow \lambda$ está en R_L si y solo si $\lambda \in L(G)$ y $A = S_L$.

Ejercicio

Encuentre G_L para las siguientes gramáticas:

1. $G = (V, \Sigma, R, S)$, donde:

   $V = \{S, A, B, C, a, b, c\}$

   $\Sigma = \{a, b, c\}$

   $R = \{S \rightarrow ACA,$

   $A \rightarrow aAa,$

   $A \rightarrow B,$

   $A \rightarrow C,$

   $B \rightarrow bB,$

   $B \rightarrow b,$

   $B \rightarrow \lambda,$

   $C \rightarrow cC,$

   $C \rightarrow \lambda\}$

2. $G = (V, \Sigma, R, S)$, donde:

   $V = \{S, B, a, b\}$

   $\Sigma = \{a, b\}$

   $R = \{S \rightarrow SaB,$

   $S \rightarrow aB,$

B → *b*B,

B → λ}

3. G = (V, Σ, R, S), donde:

V = {S, A, B, C, *a*, *b*, *c*}

Σ = {*a*, *b*, *c*}

R = {S → *a*S,

S → AB,

S → AC,

A → *a*A,

A → λ,

B → *b*B,

B → *b*S,

B → λ,

C → *c*C,

C → λ}

4.3.3 Eliminación de Reglas de Cadena

Cuando aplicamos la regla de la forma A → B no se incrementa la longitud de la cadena derivada y tampoco se producen símbolos terminales adicionales, sino que simplemente se cambia de nombre a una variable, Las reglas de esta forma son llamadas ***reglas de cadena***. El remover las reglas

de cadena requiere la adición de reglas que permitan a la gramática revisada generar las mismas cadenas. Esto se obtiene, de la misma manera que con las reglas lambda, reemplazando en las reglas las derivaciones posibles que se tendrían al usar las reglas de cadena.

La regla de cadena A → B indica que cualquier cadena que puede derivarse a partir de la variable B también puede derivarse a partir de A. Por lo tanto, la regla de cadena puede ser eliminada agregando reglas A que directamente generen las mismas cadenas que las reglas B, es decir que una regla A → *w* puede ser agregada por cada regla B → *w* y eliminar la regla de cadena.

Por ejemplo, si tenemos las siguientes reglas en una gramática:

A → *a*A

A → *a*

A → B

B → *b*B

B → *b*

B → C

Y añadimos las reglas anteriormente indicadas, obtenemos:

A → *a*A

A → *a*

A → *b*B

A → *b*

A → C

B → *b*B

B → *b*

B → C

pero esto generó una nueva regla de cadena, que puede ser eliminada repitiendo el procedimiento anterior, o podemos primero encontrar la ***cadena de símbolos equivalentes***, para luego eliminar las reglas de cadena.

Podemos encontrar esta cadena de símbolos equivalentes, utilizando el siguiente algoritmo:

Construcción del Conjunto CADENA(A)

Entrada: Una gramática libre de contexto esencialmente no contractable $G_L = (V_L, \Sigma_L, R_L, S_L)$

1. CADENA(A) = {A}
2. BUFF = { }
3. repetir:
   - i. OTRO = CADENA(A) – BUFF
   - ii. BUFF = CADENA(A)
   - iii. para cada variable B ∈ OTRO:
     - para cada regla B → C:
       - CADENA(A) = CADENA(A) ∪ {C}

   hasta que CADENA(A) = BUFF

Ejercicio

Para las siguientes reglas de una gramática G_L encuentre CADENA(A)

S → ACA

S → CA

S → AA

S → AC

S → A

S → C

S → λ

A → *a*A*a*

A → *aa*

A → B

A → C

B → *b*B

B → *b*

C → *c*C

C → *c*

Sea $G_L = (V_L, \Sigma_L, R_L, S_L)$ una gramática libre de contexto esencialmente no contractable, entonces existe un algoritmo que construye una gramática libre de contexto equivalente G_C, que satisface:

i. $L(G_C) = L(G_L)$

ii. G_C no tiene reglas de cadena.

Las reglas A de G_C son construidas a partir del conjunto CADENA(A) y de las reglas de G_L. La regla A → *w* está en R_C si hay una variable B y una cadena *w* que satisface.

i. $B \in CADENA(A)$

ii. $B \rightarrow w \in R_L$

iii. $w \notin (V_L - \Sigma)$

Las variables, el alfabeto y el símbolo inicial de G_C son los mismos que en G_L.

Ejercicio

Construya G_C para la gramática siguiente:

G = (V, Σ, R, S), donde:

V = {S, A, B, C, *a*, *b*, *c*}

Σ = {*a*, *b*, *c*}

R = {S → ACA,

S → CA,

S → AA,

S → AC,

S → A,

S → C,

S → λ,

A → *a*A*a*,

A → *aa*,

A → B,

A → C,

B → *b*B,

B → *b*,

C → *c*C,

C → *c*}

4.3.4 Eliminación de Símbolos Inútiles

Un terminal es usado si aparece en una cadena en el lenguaje G. Una variable es usada si aparece en una derivación que iniciando con el símbolo inicial genera una cadena terminal. Es decir, la variable debe aparecer en una forma sentencial de la gramática y dicha forma sentencial debe terminar en una cadena terminal. Un símbolo que no genera cadenas terminales se dice que es un ***símbolo inútil***.

Podemos utilizar el siguiente algoritmo para hallar las variables usadas y eliminar las inútiles:

Construcción del Conjunto de Variables que Derivan Cadenas Terminales

Entrada: Una gramática libre de contexto esencialmente no contractable $G_L = (V_L, \Sigma_L, R_L, S_L)$

1. TERMINAL = {A : hay una regla A → $w \in R$ con $w \in \Sigma^*$}
2. repetir
   i. BUFF = TERMINAL
   ii. para cada variable $A \in (V - \Sigma)$:

si hay una regla A de la forma $A \rightarrow w$

y $w \in (BUFF \cup \Sigma)^*$ entonces:

$TERMINAL = TERMINAL \cup \{A\}$

hasta que $BUFF = TERMINAL$

Las variables que no se encuentran en TERMINAL son inútiles y se las puede eliminar, ya que no generan ninguna cadena del lenguaje.

Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto, entonces existe un algoritmo que construye una gramática libre de contexto $G_T = (V_T, \Sigma_T, R_T, S_T)$ que satisface:

i. $L(G_T) = L(G)$
ii. Cada variable en G_T deriva una cadena terminal.

De este modo, se obtiene una gramática sin símbolos inútiles, con las siguientes características:

$V_T = TERMINAL \cup \Sigma_T$

$R_T = \{A \rightarrow w : A \rightarrow w \in R, A \in (V_T - \Sigma_T)$ y $w \in (V_T)^* \}$

$\Sigma_T = \{a \in \Sigma : a$ ocurre en el lado derecho de $R_T \}$

Ejercicio

Para la siguiente gramática obtenga G_T

$G = (V, \Sigma, R, S)$, donde:

$V = \{S, A, B, C, D, E, F, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow AC,$

S → BS,

S → B,

A → *a*A,

A → *a*F,

B → CF,

B → *b*,

C → *c*C,

C → D,

D → *a*D,

D → BD,

D → C,

E → *a*A,

E → BSA,

F → *b*B,

F → *b*}

4.3.5 Eliminación de Variables no Alcanzables

Una gramática puede contener reglas que no son utilizadas, debido a que las variables no terminales utilizadas en el lado izquierdo de dichas reglas, no son utilizadas en derivaciones que se inician en el símbolo inicial S. Dichas variables son llamadas variables no alcanzables, y son inútiles en el gramática. Para encontrar las variables que sí son alcanzables y eliminar las otras, podemos utilizar el siguiente algoritmo:

Construcción del Conjunto de Variables Alcanzables

Entrada: Una gramática libre de contexto esencialmente no contractable $G_L = (V_L, \Sigma_L, R_L, S_L)$

1. ALCANZA = {S}
2. BUFF = { }
3. repetir
   i. OTRA = ALCANZA - BUFF
   ii. BUFF = ALCANZA
   iii. para cada variable $A \in$ OTRA:
      para cada regla $A \rightarrow w$:
         ALCANZA = {ALCANZA $\cup$ {B : B
         ocurre en el lado derecho de A y B
         $\in (V_L - \Sigma_L)$}

   hasta que ALCANZA = BUFF

Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto, entonces existe un algoritmo que construye una gramática libre de contexto $G_U = (V_U, \Sigma_U, R_U, S_U)$ que satisface:

i. $L(G_U) = L(G)$

ii. G_U no tiene símbolos inútiles.

El remover símbolos inútiles parte con la construcción de G_T a partir de G. Posteriormente se aplica el algoritmo para generar las variables de G_T que son alcanzables desde el símbolo inicial. Todas las reglas de G_T que hacen

referencia a variables no alcanzables desde S son borradas para obtener G_U:

V_U = ALCANZA $\cup$ Σ_U

R_U = {A $\rightarrow$ *w* : A $\rightarrow$ *w* $\in$ R_T, A $\in$ (V_U - Σ_U) y *w* $\in$ (V_U)*}

Σ_U = {*a* $\in$ Σ : *a* ocurre en el lado derecho de las reglas en R_U}

Ejercicio

1. Para la gramática G_T obtenida previamente, obtenga G_U
2. Para la siguiente gramática, elimine los símbolos inútiles.

   G = (V, Σ, R, S), donde:

   V = {S, A, B, *a*, *b*}

   Σ = {*a*, *b*}

   R = {S $\rightarrow$ *a*,

   S $\rightarrow$ AB,

   A $\rightarrow$ *b*}

4.3.6 Forma Normal de Chomsky

Una forma normal es descrita por un conjunto de condiciones que cada regla en la gramática debe satisfacer. La forma normal de Chomsky establece restricciones sobre la longitud y composición del lado derecho de la regla. Una gramática libre de contexto G = (V, Σ, R, S) está en la ***forma normal de Chomsky*** si cada regla tiene una de las siguiente formas:

i. $A \rightarrow BC$
ii. $A \rightarrow a$
iii. $S \rightarrow \lambda$,

donde $B, C \in (V - \{S\} - \Sigma)$.

El árbol de derivación de una gramática en la forma normal de Chomsky es un árbol binario.

Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto, entonces existe un algoritmo para construir una gramática $G' = (V', \Sigma', R', S')$ en la forma normal de Chomsky que es equivalente a G, si y solo si G cumple con:

i. El símbolo de inicio de G no es recursivo
ii. G no contiene reglas lambda diferentes a $S \rightarrow \lambda$
iii. G no contiene reglas de cadena
iv. G no contiene símbolos inútiles
v. G no contiene variables no alcanzables

Una regla satisface las condiciones previas si tiene la forma:

$S \rightarrow \lambda$,
$A \rightarrow a$ o
$A \rightarrow w$,
donde $w \in (V - \{S\})^*$ y $\text{longitud}(w) > 1$.

El conjunto R' es construido a partir de las reglas de G, de la siguiente manera:

Sea A → *w* una regla con longitud(*w*) > 1. La cadena *w* puede contener símbolos terminales y no terminales, entonces, para eliminar los símbolos terminales, se agrega un nuevo símbolo y una nueva regla, cuya función es cambiar el nombre del símbolo terminal por el de un símbolo no terminal.

Por ejemplo, si tenemos la siguiente regla en una gramática:

A → *b*D*c*F

Introducimos un nuevo símbolo no terminal por cada terminal de la regla:

A → B'DC'F

B' → *b*

C' → *c*

Luego, las reglas resultantes con más de dos símbolos no terminales en su lado derecho, debe ser divididas de modo que solo queden dos, según las condiciones de la forma normal de Chomsky, de la siguiente manera:

A → B'T_1

T_1 → DT_2

T_2 → C'F

Ejercicio

Para las siguientes gramáticas, encuentre la forma normal de Chomsky:

1. G = (V, Σ, R, S), donde:

   V = {S, A, B, C, *a*, *b*, *c*}

   Σ = {*a*, *b*, *c*}

   R = {S → *a*ABC,

S → *a*,
A → *a*A
A → *a*,
B → *bc*B,
B → *bc*,
C → *c*C,
C → *c*}

2. G = (V, Σ, R, S), donde:
V = {S, A, T, *a*, *b*, +, (,)}
Σ = {*a*, *b*, +, (,)}
R = {S → A+T,
S → *b*,
S → (A)
A → A+T,
A → *b*,
A → (A)
T → *b*,
T → (A)}

4.3.7 Eliminación de Recursividad a la Izquierda

Consideremos las derivaciones usando las reglas A → A*a* *y* *A* → *b*. La aplicación repetitiva de la regla directamente recursiva A → A*a*, que por tener el símbolo no terminal en el extremo izquierdo de la cadena de la

derecha de la regla, es una ***recursividad a la izquierda***, produce cadenas de la forma Aa^i, donde lo que se consigue es la repetición de a un número arbitrario de veces o sea a^*. La derivación de cualquier cadena generada de este modo, termina con la aplicación de la regla no recursiva $A \rightarrow b$, generando esto ba^*.

Para eliminar la recursividad a la izquierda se requiere la adición de un símbolo no terminal a la gramática. Este símbolo introduce un conjunto de reglas recursivas hacia la derecha, lo cual causa que la recursividad ocurra en el símbolo más hacia la derecha en las derivaciones.

Para eliminar la recursividad a la izquierda, las reglas A son divididas en dos categorías: las reglas recursivas y las reglas que no representan recursividad a la izquierda. Las reglas recursivas tienen la siguiente forma:

$A \rightarrow Au_1$

$A \rightarrow Au_2$

...

$A \rightarrow Au_j$

y las reglas sin recursividad tienen la forma:

$A \rightarrow v_1$

$A \rightarrow v_2$

...

$A \rightarrow v_k$

en las cuales, v_i no tiene un símbolo no terminal como primer símbolo de la cadena.

Las reglas sin recursividad no cambian. Pero se añaden otras reglas que trasladen la recursividad hacia la derecha, usando las mismas cadenas v_i con el nuevo símbolo no terminal Z, para poder eliminar las reglas con recursividad a la izquierda.

$A \rightarrow v_1$

$A \rightarrow v_2$

...

$A \rightarrow v_k$

$A \rightarrow v_1 Z$

$A \rightarrow v_2 Z$

...

$A \rightarrow v_k Z$

Si existen reglas con una secuencia de u_i's, estas son generadas por las reglas del nuevo símbolo no terminal Z, utilizando recursividad a la derecha:

$Z \rightarrow u_1 Z$

$Z \rightarrow u_2 Z$

...

$Z \rightarrow u_j Z$

$Z \rightarrow u_1$

$Z \rightarrow u_2$

...

$Z \rightarrow u_j$

Ejercicio

Elimine la recursividad a la izquierda de las siguientes gramáticas:

1. G = (V, Σ, R, S), donde:

   V = {S, *a*, *b*}

   Σ = {*a*, *b*}

   R = {S → S*a*,

   S → *b*}

2. G = (V, Σ, R, S), donde:

   V = {S, *a*, *b*, *c*}

   Σ = {*a*, *b*, *c*}

   R = {S → S*a*,

   S → S*b*,

   S → *b*

   S → *c*}

3. G = (V, Σ, R, S), donde:

   V = {S, A, *a*, *b*, *c*}

   Σ = {*a*, *b*, *c*}

   R = {S → SA,

   S → AS,

   S → *a*,

   A → *b*,

   A → *c*}

4.3.8 Forma Normal de Greibach

Una gramática libre de contexto $G = (V, \Sigma, R, S)$ está en la ***forma normal de Greibach*** si cada regla tiene una de las siguientes formas:

i. $A \rightarrow aA_1A_2\ldots A_n$

ii. $A \rightarrow a$

iii. $S \rightarrow \lambda$

donde $a \in \Sigma$ y $A_i \in (V - \{S\} - \Sigma)$ para $i = 1, 2, \ldots, n$

Sea $G = (V, \Sigma, R, S)$ una gramática libre de contexto. Entonces se pueden transformar las reglas para que cumplan con las condiciones de la forma normal de Greibach, y obtener una gramática G' equivalente. Por ejemplo, si las siguientes son reglas de una gramática:

$A \rightarrow uBv$

$B \rightarrow w_1,$

$B \rightarrow w_2,$

$\ldots,$

$B \rightarrow w_n$

Donde $u, v, w_i \in \Sigma^*$.

Para eliminar el uso de símbolos no terminales en medio de las cadenas de la derecha de las reglas, se debe añadir las reglas:

$A \rightarrow u\, w_1 v,$

$A \rightarrow u\, w_2 v,$

...

$A \rightarrow u\ w_n v$,

Consiguiendo una gramática G' = (V, Σ, R', S) equivalente a G.

Ejercicio

Para la siguiente gramática construya las formas normales de Chomsky y Greibach

1. G = (V, Σ, R, S), donde:

   V = {S, B, *a*, *b*}

   Σ = {*a*, *b*}

   R = {S → S*a*B,

   S → *ab*,

   B → *b*B,

   B → λ}

4.4 Obtención de Lenguajes a partir de Gramáticas

La obtención de un lenguaje generado a partir de una gramática, se realiza a través del análisis de todas las combinaciones posibles que se pueden obtener a partir de la gramática:

Por ejemplo, sea la gramática G, obtenemos el lenguaje que genera:

G = (V, Σ, R, S), donde:

$V = \{S, A, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow aSa,$

$S \rightarrow aBa,$

$B \rightarrow bB,$

$B \rightarrow b\}$

La regla $S \rightarrow aSa$ genera en forma recursiva un número igual de *a's* tanto al principio como al final de la cadena, hasta que se aplica la regla $S \rightarrow aBa$, por lo que las cadenas generadas tendrán por lo menos un par de *a's*. El símbolo no terminal B, genera en el medio de la cadena mediante la regla recursiva $B \rightarrow bB$, cualquier número de de *b's*, pero por lo menos una, mediante la regla $B \rightarrow b$. La gramática expresada en notación de conjuntos es $L(G) = \{a^n b^m a^n : n \geq 1, m \geq 1\}$.

Ejercicio

Para las siguientes gramáticas, obtenga el lenguaje que generan y expréselo en notación de conjuntos:

1. $G = (V, \Sigma, R, S)$, donde:

   $V = \{S, a, b\}$

   $\Sigma = \{a, b\}$

   $R = \{S \rightarrow aSb,$

   $S \rightarrow aSbb,$

   $S \rightarrow \lambda\}$

2. G = (V, Σ, R, S), donde:

   V = {S, B, *a*, *b*, *c*}

   Σ = {*a*, *b*, *c*}

   R = {S → *ab*S*c*B,

   S → λ,

   B → *b*B,

   B → *b*}

4.5 Obtención de Gramáticas a partir de Lenguajes

Existen tres metodologías para construir gramáticas:

1. Analizando el lenguaje que genera, cuando este se encuentra en notación de conjuntos.
2. Partiendo de las reglas para el símbolo inicial e ir construyendo las demás reglas hasta llegar a reglas con solo símbolos terminales a la derecha (de arriba hacia abajo).
3. Partiendo de las reglas con símbolos terminales a la derecha e irlas integrando hasta llegar a las reglas del símbolo inicial (de abajo hacia arriba).

Por ejemplo si el lenguaje L(G) = $\{a^n b^m c^m d^{2n} : n \geq 0, m > 0\}$, encontrar G. Las repeticiones de *a's* al principio de la cadena y de *d's* al final, indican que se tienen que construir reglas recursivas, en que el número de *d's* sea el doble

que el de *a's*. Para las *b's* y *c's*, las cuales son generadas en igual número y repetitivamente, se debe construir otra regla recursiva. Por tanto, para las *a's* y *d's* tendremos la regla:

S → *a*S*dd*,

Ahora requerimos de un punto de salida y como $n \geq 0$, indica que las *a's* y *d's* no necesariamente tienen que estar presentes, pero como $m > 0$, la otra parte de las cadenas sí debe estar presente. Entonces tendremos una regla de la forma:

S → A.

Para tratar la parte restante tendremos dos reglas, una recursiva y otra que indica el punto de salida:

A → *b*A*c* y

A → *bc*.

Ejercicios

1. Una cadena w es palíndroma si $w = w^R$. Construya una gramática que acepte las cadenas palíndromas generadas a partir de {*a*, *b*}.

2. Construya una gramática para expresiones donde se involucre multiplicación, división suma y resta.

3. Construya una gramática para expresiones, donde además de involucrar las operaciones antes mencionadas, se involucre operaciones lógicas y de relación.

4. Construya una gramática para LEX, donde los terminales son: Conjuntos, Definiciones, Operadores, Acciones, Comentarios y Caracteres.

4.6 Gramáticas Regulares y Autómatas Finitos

4.6.1 Gramáticas Regulares

Una ***gramática regular*** es una gramática libre de contexto, en la que cada regla tiene una de las siguientes formas:

i. $A \rightarrow w$

ii. $A \rightarrow wB$

iii. $A \rightarrow \lambda$,

donde $A, B \in (V - \Sigma)$ y $w \in \Sigma^*$. Un lenguaje es regular, si puede ser generado por una gramática regular.

Ejercicios

1. Indique cuáles de las siguientes gramáticas son regulares y cuales no.
   a. $G = (V, \Sigma, R, S)$, donde:

      $V = \{S, A, B, a, b\}$

      $\Sigma = \{a, b\}$

      $R = \{S \rightarrow AB,$

A → *a*A,

A → *a*,

B → *b*B,

B → λ}

b. G = (V, Σ, R, S), donde:

V = {S, B, *a*, *b*}

Σ = {*a*, *b*}

R = {S → *a*S,

S → *a*B,

B → *b*B,

B → λ}

c. Construya el árbol de derivación para la cadena *ababaaa* para ambas gramáticas.

2. Construya una gramática regular que genere el lenguaje de la gramática:

G = (V, Σ, R, S), donde:

V = {S, B, *a*, *b*}

Σ = {*a*, *b*}

R = {S → *ab*SA,

S → λ,

A → A*a*,

A → λ}

4.6.2 Gramáticas Regulares y Autómatas Finitos

Sea G = (V, Σ, R, S) una gramática regular, definimos el AFND M = (K, Σ, Δ, *s*, F) como sigue:

i. K = (V - Σ) ∪ {f}

ii. (A, *a*, B) ∈ Δ si A → *a*B ∈ R

(A, *a*, f) ∈ Δ si A → *a* ∈ R

iii. F = {A : A → λ ∈ R} ∪ {f}

Y encontramos que L(M) = L(G).

Ejercicios

1. Dadas las siguientes gramáticas, obtenga un AFND y una expresión regular que denote el lenguaje.

a. G = (V, Σ, R, S), donde:

V = {S, A, *a*, *b*}

Σ = {*a*, *b*}

R = {S → *a*S,

S → *a*A,

A → *b*A,

A → *b*}

b. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, B, Z, a, b\}$

$\Sigma = \{a, b\}$

$R = \{S \rightarrow aS,$

$S \rightarrow bB,$

$S \rightarrow aZ,$

$B \rightarrow bB,$

$B \rightarrow \lambda,$

$Z \rightarrow \lambda\}$

c. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, B, a, b\}$

$\Sigma = \{a, b\}$

$R = \{S \rightarrow aS,$

$S \rightarrow bB,$

$S \rightarrow a,$

$B \rightarrow bB,$

$B \rightarrow \lambda\}$

d. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, B, C, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow bS,$

$S \rightarrow cS,$

$S \rightarrow aB,$

S → λ,

B → *a*B,

B → *c*S,

B → *b*C,

B → λ,

C → *a*B,

C → *b*S,

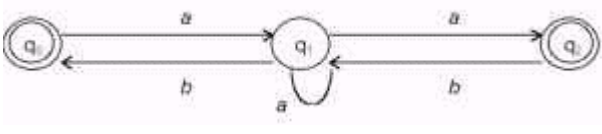
C → λ}

2. Para las siguientes expresiones regulares encuentre una gramática que genere el lenguaje denotado por dicha expresión regular:

   a) $(b \cup ab)^* (a \cup \lambda)$

   b) $(ab)^*ba$

3. Para el siguiente AFND construya una gramática regular y escriba una expresión regular para L(M)



4.7 Ambigüedad

Sea G = (V, Σ, R, S) una gramática libre de contexto. Una cadena *w* está en L(G) si y solo si, existe una derivación más hacia la izquierda de *w* a partir de S. Una gramática libre de contexto G es una ***gramática ambigua*** si hay una cadena $w \in L(G)$ que pueda ser derivada de dos o más formas distintas utilizando derivaciones más hacia la izquierda. Una gramática que no es ambigua es denominada ***gramática sin ambigüedad***.

Ejercicios

1. Sea la gramática

   G = (V, Σ, R, S), donde:

   V = {S, *a*}

   Σ = {*a*}

   R = {S → *a*S,

   S → S*a*,

   S → *a*}

   a. Determine si *aa* es elemento de L(G).
   b. Determine si la gramática es ambigua.
   c. Escriba una gramática no ambigua que reconozca L(G).

2. Determine si la siguiente gramática es ambigua o no.

G = (V, Σ, R, S), donde:

V = {S, *if EXP*, *then INS*, *else INS*}

Σ = {*if EXP*, *then INS*, *else INS*}

R = {S → *if EXP then INS*,

S → *if EXP then INS else INS*}

3. Determine si la siguiente gramática es ambigua

G = (V, Σ, R, E), donde:

V = {E, id, +, ∗, (,), *a*, *b*, *c*, *d*}

Σ = {+, ∗, (,), *a*, *b*, *c*, *d*}

R = {E → E+E,

E → E∗E,

E → (E),

E → id,

id → *a*,

id → *b*,

id → *c*,

id → *d*}

pruebe con *a*+*b*∗*c*+(*d*∗*a*) y escriba una gramática no ambigua.

4.8 Autómatas de Pila (Pushdown)

Los lenguajes regulares son aquellos lenguajes generados por gramáticas regulares y aceptados por un autómata finito. Los lenguajes no regulares necesitan una memoria adicional en forma de pila. Un autómata de pila es una máquina de estados finitos que incluye una pila.

Un ***autómata de pila*** es un sextuple $(K, \Sigma, \Gamma, \Delta, s, F)$, donde:

K es un conjunto finito de ***estados***

Σ es un conjunto finito de símbolos llamado el ***alfabeto de entrada***

Γ es un conjunto finito de símbolos llamado ***alfabeto de pila***

s es el ***estado inicial***

$F \subseteq K$ es el conjunto de ***estados finales***

Δ es una relación de transición de $((K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*))$.

Un AP tiene dos alfabetos: un alfabeto de entrada Σ a partir del cual las cadenas de entrada son construidas y un alfabeto de pila cuyos elementos son almacenados en una pila. Las características de la pila son las siguientes:

- La pila es representada como una cadena de elementos.
- El elemento en el tope de la pila es el símbolo más hacia la izquierda en la cadena, representando al último símbolo en entrar a la pila.
- Las cadenas de símbolos de la pila son representadas por letras griegas
- Una pila vacía es denotada con λ.

Las transiciones son de la forma:

$$((q,a,\alpha), (p,\beta)) \in \Delta$$

y representan la operación de un autómata de pila al leer el símbolo a, asegurándose que la cadena en el tope de la pila es α y cambiándola por β.

Un AP puede ser representado por un diagrama de estados, donde las etiquetas en el arco indican el símbolo de entrada y la operación en la pila. La transición $((q,a,\alpha), (p,\beta))$ es representada como a, α/β, donde el símbolo / indica el remplazo de α por β en el tope de la pila.

Las dos operaciones de pila, ***push*** (meter), y ***pop*** (sacar), pueden representarse como:

$((q, \lambda, a), (q, \lambda))$ pop (sacar)

$((q, \lambda, \lambda), (q, a))$ push (meter)

En estos casos, como el símbolo de entrada es λ, la transición no procesa un símbolo de entrada, por lo tanto, la transición saca o mete el símbolo de la pila a sin alterar el estado o la entrada.

También podemos tener las transiciones:

$((q, a, a), (q, \lambda))$ pop (sacar)

$((q, a, \lambda), (q, a))$ push (meter)

En estos casos, como el símbolo de entrada es a, la transición mete en la pila el símbolo leído, o busca en el tope de la pila el mismo símbolo que lee para sacarlo de la pila.

La transición:

$$((q, a, \lambda), (q, \lambda))$$

corresponde a la transición de un autómata finito, porque su aplicación depende exclusivamente del estado y símbolo de entrada.

La ***configuración*** de un AP está representada por un triple:

$$(q, w, \alpha)$$

donde q representa el estado actual, w es la cadena por leer, y α es la cadena en el tope de la pila.

La función ***paso de configuración***:

$$(q_i, w, \alpha) \vdash (q_j, v, \beta)$$

se da, solo si:

$$w = uv \text{ y}$$

$$((q_i, u, \alpha), (q_j, \beta)) \in \Delta$$

Entonces, el símbolo $\vdash$ representa una secuencia de transiciones. La ***cerradura reflexiva y transitiva*** del paso de configuración se representa por $\vdash^*$. La computación de un AP es una secuencia de configuraciones comenzando la máquina con el estado inicial y la pila vacía. Una cadena w es aceptada por un autómata de pila si:

$$(s, w, \lambda) \vdash^* (q, \lambda, \lambda); q \in F$$

si comenzando en el estado inicial s, con la cadena w completa por procesar y con la pila vacía, alcanza un estado final, leyendo toda la cadena y con la pila vacía. El ***lenguaje*** de M, denotado L(M), es el conjunto de cadenas aceptadas por M.

Una cadena es rechazada por el autómata de pila en los siguientes casos:

- Si una operación procesa toda la cadena de entrada y termina en un estado de no aceptación.
- Si hay una operación que no completa el procesamiento de la cadena de entrada.
- Una operación que procesa la cadena de entrada por completo pero que no termina de vaciar la pila.

Por ejemplo, consideremos un AP que acepta el lenguaje $\{a^i b^i : i \geq 0\}$, de tal forma que el símbolo de entrada a, causa que a entre a la pila y el símbolo de entrada b saque un elemento del tope de la pila. El lenguaje genera todas las cadenas de un número de a´s seguido por un número igual de b´s. Entonces si $w = aabb$, el AP es el siguiente:

$M = (K, \Sigma, \Gamma, \Delta, s, F)$

$K = \{q_0, q_1\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a\}$

$F = \{q_0, q_1\}$

$\Delta = \{((q_0, a, \lambda), (q_0, a)),$

$((q_0, b, a), (q_1, \lambda))\}$

Las transiciones para la cadena $aabb$ son:

$(q_0, aabb, \lambda) \vdash (q_0, abb, a) \vdash (q_0, bb, aa) \vdash (q_1, b, a) \vdash (q_1, \lambda, \lambda)$

ya que *q1* es estado final, la cadena se lee completa y la pila queda vacía, la cadena *aabb* es aceptada.

Un AP es ***no determinístico*** si hay más de una transición aplicable para la combinación estado, símbolo de entrada y tope de la pila.

Ejercicio

Sea el AP que acepta el lenguaje $\{wcw^R : w \in \{a, b\}^*\}$. Los símbolos del pila son *a* y *b*, los cuales indican la lectura de los símbolos de entrada *a* y *b,* respectivamente. Entonces:

$M = (K, \Sigma, \Gamma, \Delta, s, \mathbf{F})$

$K = \{q_0, q_1\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{a, b\}$

$F = \{q_1\}$

$\Delta = \{((q_0, a, \lambda), (q_0, a)),$

$((q_0, b, \lambda), (q_o, b)),$

$((q_0, c, \lambda), (q_1, \lambda)),$

$((q_1, a, a), (q_1, \lambda)),$

$((q_1, b, b), (q_1, \lambda))\}$

a) Construya el diagrama de estados de M

b) Escriba los pasos de configuración para la cadena *abcba*

c) Muestre que *aca* y *bcb* $\in$ L(M).

d) Muestre que *acb* y *bca* $\notin$ L(M).

Un AP es ***determinístico*** si hay una sola transición que es aplicable para cada combinación de estado, símbolo de entrada y tope del pila. Dos transiciones $((q_i, u, \alpha), (q_j, \beta)) \in \Delta$ y $((q_i, v, \chi), (q_k, \gamma)) \varepsilon \Delta$ son llamadas ***compatibles*** si cualquiera de las siguientes condiciones es satisfecha:

$u = v$ y $\alpha = \chi$

$u = v$ y $\alpha = \lambda$ o $\chi = \lambda$

$\alpha = \chi$ y $u = \lambda$ o $v = \lambda$

$u = \lambda$ o $v = \lambda$ y $\alpha = \lambda$ o $\chi = \lambda$

Ejercicios

1. Sea $L = \{a^i : i \geq 0\} \cup \{a^i b^i : i \geq 0\}$ un lenguaje que acepta *a's* o igual número de *a's* y *b's*, con las *b's* siguiendo las *a's*. La pila del AP que acepta L lleva un contador de *a's* procesadas hasta que una *b* es encontrada o toda la cadena de entrada es procesada y luego se controla que el número de *b's* que se procese sea igual al número de *a's* en la pila.

$M = (K, \Sigma, \Gamma, \Delta, s, F)$

$K = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a\}$

$F = \{q_1, q_2\}$

$\Delta = \{((q_0, a, \lambda), (q_0, a)),$

$((q_0, b, a), (q_1, \lambda)),$

$((q_0, \lambda, \lambda), (q_2, \lambda)),$

$((q_1, b, a), (q_1, \lambda)),$

$((q_2, \lambda, a), (q_2, \lambda))\}$

a) Escriba el diagrama de estados de M

b) Escriba los pasos de configuración para las cadena *aaaa* y *aabb*

2. La máquina M acepta el lenguaje $L = \{a^i b^i : i > 0\}$ por estado final y por pila vacía.
   a) Dibuje un diagrama de transición de estados de un AP que acepte L por pila vacía.
   b) Dibuje un diagrama de transición de estados de un AP que acepte L por estado final.

4.8.1 Autómatas de Pila y Lenguajes Libres de Contexto

Existe una correspondencia entre las operaciones de un autómata de pila como reconocedor de un lenguaje libre de contexto y las derivaciones en una gramática libre de contexto. Para poder ver como un AP puede reconocer una gramática libre de contexto consideramos lo siguiente:

- Las reglas de la gramática son usadas para generar las transiciones de un AP equivalente.

- Sea L un lenguaje libre de contexto y G una gramática en la forma normal de Greibach donde L(G) = L.
- Las reglas de G, excepto $S \rightarrow \lambda$, tienen la forma $A \rightarrow aA_1A_2 \ldots A_n$.
- En una derivación de más hacia la izquierda, las variable A_i son procesadas de izquierda a derecha, metiendo a la pila las variables en el orden requerido por las derivaciones.

Por ejemplo, si tenemos la siguiente gramática en la forma normal de Greibach:

G = (V, Σ, R, S), donde:

V = {S, A, B, *a*, *b*}

Σ = {*a*, *b*}

R = {S → *a*AB,

S → *a*B,

A → *a*AB,

A → *a*B,

B → *b*}

entonces el AP tiene dos estados, un estado inicial q_0 y un estado final q_1. Una regla de la forma $S \rightarrow aA_1A_2 \ldots A_n$ genera una transición que procesa el símbolo terminal *a*, mete los símbolos no terminales $A_1A_2 \ldots A_n$ a la pila y se llega al estado q_1. El resto de las transiciones usan el símbolo de entrada y la cadena en el tope de la pila para determinar la transición apropiada. Siendo así, la relación de transición del AP para la gramática G es:

Δ = {((*q*0, *a*, *λ*), (*q*1, AB)),

((*q*0, *a*, *λ*), (*q*1, B)),

$((q_1, a, A), (q_1, AB)),$

$((q_1, a, A), (q_1, B)),$

$((q_1, b, B), (q_1, \lambda))\}$

Aplicando la secuencia de pasos de configuración y las derivaciones para la cadena *aaabbb*, podemos ver la relación que existe:

$S \Rightarrow aAB \qquad (q_0, aaabbb, \lambda) \vdash (q_1, aabbb, AB)$

$\Rightarrow aaABB \qquad \vdash (q_1, abbb, ABB)$

$\Rightarrow aaaBBB \qquad \vdash (q_1, bbb, BBB)$

$\Rightarrow aaabBB \qquad \vdash (q_1, bb, BB)$

$\Rightarrow aaabbB \qquad \vdash (q_1, b, B)$

$\Rightarrow aaabbb \qquad \vdash (q_1, \lambda, \lambda)$

Sea $G = (V, \Sigma, R, S)$ una gramática en la forma normal de Greibach que genera L. Un AP M, que reconozca el mismo lenguaje $L(G)$ generado por G, con estado inicial q_0, está definido por:

$M = (K, \Sigma, \Gamma, \Delta, s, F)$

$K = \{q_0, q_1\}$

$\Sigma = \Sigma$

$\Gamma = V - \{S\} - \Sigma$

$F = \{q_1\}$

con transiciones:

$\Delta = \{((q_0, a, \lambda), (q_1, w)) : S \rightarrow aw \in R$

$((q_1, a, A), (q_1, w)) : A \rightarrow aw \in R$ y $A \in V - \{S\} - \Sigma$

$((q_0, \lambda, \lambda), (q_1, \lambda))$ Si $S \rightarrow \lambda \in R\}$

Ejercicio

Escriba un AP equivalente para las siguientes gramáticas.

a. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, A, a, b\}$

$\Sigma = \{a, b\}$

$R = \{S \rightarrow aAA,$

$A \rightarrow aS,$

$A \rightarrow bS,$

$A \rightarrow a\}$

b. $G = (V, \Sigma, R, S)$, donde:

$V = \{S, A, B, a, b, c\}$

$\Sigma = \{a, b, c\}$

$R = \{S \rightarrow aABA,$

$S \rightarrow aBB,$

$A \rightarrow bA,$

$A \rightarrow b,$

$B \rightarrow cB,$

$B \rightarrow c\}$

Para ver como a partir de un AP podemos construir una gramática libre de contexto donde $L(M)$, el lenguaje aceptado por el AP es generado por la gramática G, es decir que $L(M) = L(G)$, tomamos lo siguiente:

- Las reglas de una gramática libre de contexto son construidas a partir de las transiciones del autómata.
- La gramática es diseñada de tal forma que la aplicación de una regla corresponde a una transición en el APD.

Sea $M = (K, \Sigma, \Gamma, \Delta, s, F)$ una AP, entonces una AP' con relación de transición Δ' es construido a partir de M adicionando transiciones a Δ en base a:

Si $((q_i, u, \lambda), (q_j, \lambda)) \in \Delta$, entonces $((q_i, u, a), (q_j, a)) \in \Delta'$ para cualquier $a \in \Gamma$.

Si $((q_i, u, \lambda), (q_j, b)) \in \Delta$, entonces $((q_i, u, a), (q_j, ba) \in \Delta$ para cualquier $a \in \Gamma$.

La interpretación de las reglas es que una transición de M que no elimina un símbolo de la pila puede ser considerada como que inicialmente saca un elemento de la pila y posteriormente mete a la pila el mismo símbolo. Cualquier cadena aceptada por pasos de configuración, utilizando una nueva transición, es aceptada por la transición original, es decir, $L(M) = L(M')$.

Una gramática $G = (V, \Sigma, R, S)$ puede ser construida a partir de las transiciones de M'. El alfabeto de G es el alfabeto de entrada de M'. Los

símbolos no terminales de G consisten del símbolo inicial S y los triples ordenados de la forma $\langle q_i, a, q_j \rangle$, donde las q's son estados de M' y $a \in \Gamma \cup \{\lambda\}$. El triple ordenado $\langle q_i, a, q_j \rangle$ representa un paso de configuración que comienza en el estado q_i, termina en q_j y elimina el símbolo a de la pila. Las reglas de G son construidas como sigue:

1. $S \rightarrow \langle q_i, \lambda, q_j \rangle$ para cada $q_j \in F$.
2. Cada transición $((q_i, u, a), (q_j, b)) \in \Delta$, donde $a \in \Gamma \cup \{\lambda\}$, genera el conjunto de reglas:

   $\{ \langle q_i, a, q_k \rangle \rightarrow u \langle q_j, b, q_k \rangle : q_k \in K\}$.

3. Cada transición $((q_i, u, a), (q_j, ba)) \in \Delta$, donde $a \in \Gamma$, genera el conjunto de reglas:

   $\{\langle q_i, a, q_k \rangle \rightarrow u \langle q_j, b, q_n \rangle \langle q_n, a, q_k \rangle : q_k, q_n \in K\}$.

4. Para cada estado $q_k \in K$,:

   $\{\langle q_k, \lambda, q_k \rangle \rightarrow \lambda\}$.

Una derivación comienza con una regla de tipo 1 cuyo lado derecho representa una operación que comienza en el estado q_0, termina en un estado final y queda la pila vacía. Las reglas de tipo 2 y 3 indican la acción de la máquina, donde las reglas de tipo 3 corresponden a las transiciones aumentadas de M'. Durante la operación, estas transiciones incrementan el tamaño de la pila, por lo que el efecto correspondiente de la regla es introducir una variable adicional en la derivación. Las reglas de tipo 4 son utilizadas para terminar las derivaciones. La regla $\langle q_k, \lambda, q_k \rangle \rightarrow \lambda$ representa una transición del estado q_k a sí mismo y no altera la pila.

Por ejemplo, construimos una gramática G a partir del AP, donde el lenguaje L(M) es el conjunto $\{a^n c b^n : n \geq 0\}$.

M = (K, Σ, Γ, Δ, *s*, F)

K = $\{q_0, q_1\}$

Σ = {*a, b, c*}

Γ = {*a*}

F = $\{q_1\}$

Δ = $\{((q_0, a, \lambda), (q_0, a)),$
$((q_0, c, \lambda), (q_1, \lambda)),$
$((q_1, b, a), (q_1, \lambda))\}$

Debido a que $((q_0, a, \lambda), (q_0, a)) \in \Delta$, entonces por la regla (2) tenemos que agregar $((q_0, a, a), (q_0, aa))$ en Δ y como $((q_0, c, \lambda), (q_1, \lambda)) \in \Delta$ por la regla (1), agregamos $((q_0, c, a), (q_1, a))$ en Δ. Sólo se agrega una transición para cada una, debido a que Γ solo contiene a *a*. Al agregar estas transiciones a M obtenemos M' y al tener M' podemos obtener G aplicando las reglas 1 a 4 para cada transición, como sigue:

Transición	**Reglas**	**Tipo aplicado**
$((q_0, a, \lambda), (q_0, a))$	$S \rightarrow \langle q_0, \lambda, q_1 \rangle$ $\langle q_0, \lambda, q_0 \rangle \rightarrow a \langle q_0, a, q_0 \rangle$ $\langle q_0, \lambda, q_1 \rangle \rightarrow a \langle q_0, a, q_1 \rangle$	1 2

$((q_0, a, a), (q_0, aa))$	$\langle q_0, a, q_0\rangle \rightarrow a \langle q_0, a, q_0\rangle \langle q_0, a, q_0\rangle$ $\langle q_0, a, q_1\rangle \rightarrow a \langle q_0, a, q_0\rangle \langle q_0, a, q_1\rangle$ $\langle q_0, a, q_0\rangle \rightarrow a \langle q_0, a, q_1\rangle \langle q_1, a, q_0\rangle$ $\langle q_0, a, q_1\rangle \rightarrow a \langle q_0, a, q_1\rangle \langle q_1, a, q_1\rangle$	3
$((q_0, c, \lambda), (q_1, \lambda))$	$\langle q_0, \lambda, q_0\rangle \rightarrow c \langle q_1, \lambda, q_0\rangle$ $\langle q_0, \lambda, q_1\rangle \rightarrow c \langle q_1, \lambda, q_1\rangle$	2
$((q_0, c, a), (q_1, a))$	$\langle q_0, a, q_0\rangle \rightarrow c \langle q_1, a, q_0\rangle$ $\langle q_0, a, q_1\rangle \rightarrow c \langle q_1, a, q_1\rangle$	2
$((q_1, b, a), (q_1, \lambda))$	$\langle q_1, a, q_0\rangle \rightarrow b \langle q_1, \lambda, q_0\rangle$ $\langle q_1, a, q_1\rangle \rightarrow b \langle q_1, \lambda, q_1\rangle$	2
	$\langle q_0, \lambda, q_0\rangle \rightarrow \lambda$ $\langle q_1, \lambda, q_1\rangle \rightarrow \lambda$	4

Si escribimos los pasos de configuración para la cadena *aacbb*:

$(q_0, aacbb, \lambda) \vdash (q_0, acbb, a)$

$\vdash (q_0, cbb, aa)$

$\vdash (q_1, bb, aa)$

$\vdash (q_1, b, a)$

$\vdash (q_1, \lambda, \lambda)$

vemos que la cadena *aacbb* pertenece a M. Por otro lado:

$S \Rightarrow \langle q_0, \lambda, q_1 \rangle$

$\Rightarrow a \langle q_0, a, q_1 \rangle$

$\Rightarrow aa \langle q_0, a, q_1 \rangle \langle q_1, a, q_1 \rangle$

$\Rightarrow aac \langle q_1, a, q_1 \rangle \langle q_1, a, q_1 \rangle$

$\Rightarrow aacb \langle q_1, \lambda, q_1 \rangle \langle q_1, a, q_1 \rangle$

$\Rightarrow aacb \langle q_1, a, q_1 \rangle$

$\Rightarrow aacbb \langle q_1, \lambda, q_1 \rangle$

$\Rightarrow aacbb$

BIBLIOGRAFIA

- BROOKSHEAR J. G. **Teoría de la Computación, Lenguajes Formales, Autómatas y Complejidad** *Addison Wesley - EEUU - 1993*

- HOPCROFT John E. & ULLMAN Jeffrey D. **Formal Languages and their Relation to Automata** *Addison Wesley - EEUU - 1979*

- HOPCROFT John E. & ULLMAN Jeffrey D. **Introducción a la teoría de Autómatas, Lenguajes y Computación** *CECSA - México - 1995*

- HOPCROFT John E. & ULLMAN Jeffrey D. **Introduction to Automata Theory, Languages and Computation** *Addison Wesley - EEUU - 1979*

- KELLEY Dean **Introducción a la Teoría de Autómatas y Lenguajes Formales** *Prentice - Hall - España - 1995*

- LEWIS Harry & PAPADIMITRIOU Christos H. **Elements of the Theory of Computation** *Prentice Hall - EEUU - 1981*

Martha Fernández Montaño

Amante tanto de las artes como de las ciencias, se inclinó hacia el fascinante mundo de las computadoras desde una temprana edad. Al finalizar su formación en piano, viajó a Estados Unidos para estudiar en LSU (Louisiana State University), donde obtuvo su B.S. en Ciencias de Computación. Al regresar a su país natal, Bolivia, obtuvo su maestría en Auditoría y Control Financiero en el programa de MpD (Maestrías para el Desarrollo de UCB-HIID (Universidad Católica Boliviana y Harvard Institute for International Development); y trabajó como Ingeniero de Sistemas en IBM Bolivia y luego como Gerente Subregional de Sistemas (Sudamérica) en Naciones Unidas.

Después de encontrar su pasión en la transmisión de conocimientos, obtuvo un diplomado en Educación Superior de UDABOL e inició su feliz carrera como docente pregrado y postgrado. Más tarde, involucrada en el campo de la auditoría y seguridad de la información, obtuvo diversas certificaciones internacionales en Auditoría y TI tanto de ISACA como de IIA (Institute of Internal Auditors).

Actualmente, aún impartiendo clases como docente universitaria, es madre a tiempo completo y también trabaja como consultora de Auditoría, Seguridad y Gobernabilidad de TI.

La Salle Universidad Bolivia

Av. Jorge Carrasco esq. Las Palmas
Nº 450, Bolognia

www.ulasalle.edu.bo

ISBN: 978-99974-63-09-8

9 789997 463098