Tipos de Lenguajes Formales

Nivardo Ibarra Florencio
Instituto Tecnológico de Chilpancingo
Centro de Ciencias de la Complejidad (C3)
Universidad Nacional Autónoma de México
nivardo_end_fin@hotmail.com

Agosto de 2011

Resumen

En el presente artículo se analizan los 4 tipos de lenguajes formales propuestos en la jerarquía de N. Chomsky, que abarca desde el conjunto de los *lenguajes regulares* hasta los *lenguajes recursivos*, dichos lenguajes son descritos mediante dos representaciones: a) *expresiones* y b)*gramáticas*; particularmente, enfocándose en la definición formal de cada uno de los lenguajes, sus propiedades, sus limitaciones y a que lenguaje(s) incluye(n) determinado tipo de lenguaje. Por otra parte se analiza la equivalencia entre las expresiones y gramáticas del lenguaje. Además se ofrece una introducción a los autómatas celulares, dando a conocer las propiedades y características de éstos, y finalmente se presenta un caso de estudio utilizando las reglas 2c22.

Contenido

1. Introducción

En los últimos 20 años se ha visto la aparición de un gran número de textos en los temas de los *lenguajes formales* y *autómatas* [3]. Esto indica la importancia y riqueza que el tema tiene. A pesar de la gran importancia de estos tópicos, este artículo esta enfocado al estudio de los *lenguajes formales*.

Por tanto en el presente artículo tratamos de explicar los lenguajes formales proporcionando inicialmente los conceptos básicos con el propósito de comprender el concepto y posteriormente se definen los lenguajes de manera formal y con ello tener una mejor adquisición de dicho conocimiento.

El presente artículo está organizado de la siguiente manera: la sección 1 ofrece una introducción general a los tipos de lenguajes formales que existen; en la sección 2, se abordan los antecedentes de la *teoría de la computación*, destacando los acontecimientos más relevantes de ésta; en la sección 3, se describen los *lenguajes regulares* los cuales se analizan sus respectivas *expresiones y gramáticas regulares*; en la sección 4, se analizan los *lenguajes libres de contexto* y sus respectivas *gramáticas libres de contexto*; en la sección 5, se describen los lenguajes que meramente se refieren a los *lenguajes sensibles del contexto*, los cuales se encuentran entre los *lenguajes libres de contexto* y los *lenguajes recursivos*, de igual manera analizando su *gramática sensible del contexto*; en la sección 6, se realiza el estudio de los *lenguajes recursivos* los cuales se subdividen en: *recursivos y recursivamente enumerables*, que son descritos por las *gramáticas no restringidas*; en la sección 7, se presenta el estudio de los autómatas celulares y algunas definiciones y elementos de la computación no convencional; en la sección 8, se analiza un caso de estudio acerca de la propagación de ondas bajo determinados patrones, utilizando las reglas 2c22; finalmente en la sección 9, exponemos las conclusiones.

2. Antecedentes

La *teoría de la computación* trata con *modelos de cálculo abstractos* que describen con distintos grados de precisión las diferentes partes y tipos de computadoras. Pero estos modelos no se usan para describir detalles prácticos del hardware de una determinada computadora, sino que más bien se ocupan de cuestiones abstractas sobre la capacidad de las computadoras en general [5].

La historia de la teoría de la computación es bastante interesante. Se ha desarrollado gracias a la confluencia, por afortunadas coincidencias, de distintos campos de conocimiento y descubrimientos (fundamentalmente matemáticos) realizados a principios del siglo XX [5].

Bajo el nombre *teoría de la computación* se conjuntan una gama de materias que conforman los fundamentos teóricos, como son: *teoría de autómatas, teoría de los lenguajes formales, computabilidad y complejidad algorítmica.*

2.1. Computabilidad

Los pioneros en éste tópico sin duda alguna fueron: Kurt Gödel, Alonzo Church, Mil Post, Alan Turing y Stephen Kleene, los cuales se fundamentan en la *lógica matemática*. Apenas iniciando el siglo XX, los matemáticos estaban a punto de realizar importantes descubrimientos, para lo cual en los 40 años siguientes estaban destinados a dar una nueva interpretación matemática y estos

logros tuvieron consecuencias que afectaron de una manera inesperada y sorprendente al campo de las *ciencias de la computación*, que aún no había nacido en ese entonces.

Al iniciar el siglo XX se empezó a investigar un dilema. George Cantor (1845-1918), había inventado por entonces la *teoría de conjuntos*, en su descubrimiento se encontró con algunas paradojas inquietantes, como las siguientes:

- Como que hay "infinitos" de distinto tamaño.
- Hay algún conjunto que sea mayor que el conjunto universal.

Estas y muchas otras paradojas dejó una gran inquietud a los matemáticos.

El punto de partida fue las cuestiones que formuló David Hilbert [5] (1862-1943) en un congreso internacional de matemáticos, en el año de 1928[1], dichas cuestiones son las siguientes:

1. ¿Son *completas* las matemáticas? en el sentido de que pueda probarse o no cada aseveración matemática.
2. ¿Son las matemáticas *consistentes*? en el sentido de que no pueda probarse simultáneamente una aseveración y su negación.
3. ¿Son las matemáticas *decidibles*? en el sentido de que exista un método definido que se pueda aplicar a cualquier aseveración matemática y que determine si dicha aseveración es cierta o falsa.

La meta de Hilbert era crear una sistema axiomático lógico-matemático *completo y consistente*, del cual podrían deducirse todas las matemáticas, esto es, cualquier teorema matemático se podría derivar a través de este sistema, aplicando una serie de pasos finitos, es decir, con lo que actualmente lo conocemos como un proceso algorítmico o computacional, con el objetivo de demostrar la verdad o falsedad de cualquier teorema en el sistema formal. A este problema le llamó *Entscheidungsproblem* (problema de decisión).

Sin embargo, en la década de 1930 se efectuaron una serie de investigaciones que mostraban que el sistema de Hilbert era imposible de realizar, en otras palabras su problema no tenía solución. Las primeras contradicciones para Hilbert aparecen en 1931 hechas por Gödel (1906-1978) y su *teorema de incompletitud*: "no es posible demostrar la consistencia de la Aritmética a partir de sus propios métodos (constructivos) de deducción" [5]. Luego Gödel volvió a demostrar que no sólo no existe un algoritmo que pueda demostrar todos los teoremas matemáticos, sino que además, no todos los resultados son demostrables. Ante dicha situación se plantean las siguientes preguntas:

- ¿Qué puede hacer la computación (sin restricciones de ningún tipo)?
- ¿Cuáles son las limitaciones inherentes a los métodos automáticos de cálculo?

A estas cuestiones pretende responder la *teoría de la computabilidad*. El primer paso en la búsqueda de las respuestas a estas preguntas está en el estudio de los modelos de computación.

Los modelos abstractos de cálculo aparecen en la década de los treinta, en los trabajos de los lógicos: Church, Gödel, Kleene, Post y Turing. Trabajos e investigaciones tuvieron un gran impacto en los orígenes para el nacimiento de la *teoría de computación*.

[1] Realizado en la ciudad de Bolonia, Emilia-Romaña, Italia

Bajo éste mismo camino Church propuso la noción de *función λ-definible* cuya función es efectivamente calculable. En 1936, Church demuestra una equivalencia entre las *funciones λ-definibles* y las *funciones recursivas* de Herbrand-Gödel y afirma que éstas funciones serían las únicas calculables por medio de un *procedimiento efectivo* a través de la tesis que lleva su nombre (*Tesis de Church*) y utilizando la noción de función λ-definible, dio ejemplos de problemas irresolubles y a su vez demostró que el *problema de decisión* era uno de esos problemas.

También casi al mismo tiempo Kleene, demuestra de forma independiente la equivalencia entre funciones λ-definibles y funciones recursivas de Herbrand-Gödel, a través del concepto de *función recursiva*.

Un matemático más que reforzó la noción de función calculable fue Turing (1912-1954). Este matemático ingles señaló que había tenido éxito en caracterizar de un modo matemáticamente preciso, por medio de sus máquinas, a las funciones calculables mediante un algoritmo (*funciones Turing-computables*), lo que se conoce como *Tesis de Turing* (1936).

Turing demostró a través de su concepto de máquina que hay problemas que no son calculables por un método definido, además de que el *problema de decisión* era de esa clase de problemas que no era posible realizarlo. Por su parte también demostró que los conceptos de función λ-definible y función calculable de Church y Kleene respectivamente son equivalentes, esto lo hizo utilizando una máquina de Turing. Por lo tanto, podemos afirmar que la Tesis de Turing y Church son equivalentes, ya que ambos demostraron lo mismo, utilizando un método o proceso diferente.

Luego por los resultados, la tesis de Church-Turing es aceptada como un axioma en la *teoría de la computación* [5], la cual ha servido como punto de partida en la investigación de los problemas que se pueden resolver mediante un procedimiento efectivo, que hoy conocemos como "algoritmo". Una de las cuestiones más estudiadas en la *teoría de la computabilidad* es la de construir programas que decidan si un determinado algoritmo poseé o no una determinada propiedad. Sería interesante responder de forma automática a cuestiones como:

- ¿Calculan los algoritmos A y B la misma función? (*Problema de la equivalencia*)
- ¿Parará el algoritmo A para una de sus entradas? (*Problema de la parada*)
- ¿Parará el algoritmo A para todas sus entradas? (*Problema de la totalidad*)
- ¿Es posible que un algoritmo A efectivamente resuelva la función f? (*Problema de la verificación*)

Poco a poco se fueron demostrando de la no computabilidad de cada una de estas cuestiones, y a su vez dejando una sensación de que casi cualquier pregunta acerca de algoritmos era no computable. A todo esto surge el *teorema de Rice*, quien afirma: "Si una propiedad de las funciones recursivas es no trivial, entonces el predicado Q asociado a P es no decidible (y el problema correspondiente no es calculable)". Es decir, nunca podremos encontrar ni programar nosotros mismos una aplicación, que pasándole por parámetros cualquier programa y una función computable, calcule correctamente si el programa computa adecuadamente dicha función. Lo que sí podremos conseguir es un programa que verifique que un programa determinado compute una función determinada.

2.2. Complejidad algorítmica

Una vez que la *teoría de la computabilidad* fuera desarrollada, fue entonces cuando se pregunta acerca de la dificultad de las funciones computables. Este es el objetivo de la parte de las ciencias de la computación conocida como "complejidad algorítmica".

Michael Oser Rabin uno de los pioneros en plantearse esta cuestión de manera explícita: ¿Qué quiere decir que una función f sea más difícil de computar que otra función g? Rabin sugirió una axiomática que fue la base para el desarrollo del estudio de *medidas de complejidad abstracta* de Lenore Blum (1967).

Otro aporte que apoyo el desarrollo de esta rama fue el artículo de Juris Hartmanis y Richard Stearns en 1965, cuyo título *On the Complexity of Algorithms* dio nombre a este cuerpo de conocimiento, mismo que introduce la noción de *medida de complejidad* definida como el tiempo de computación sobre una máquina de Turing multicinta y se demuestran los teoremas de jerarquía.

Otro logro en los inicios de este tema fue el trabajo de Alan Cobham titulado, *The Intrinsic Computational Difficulty of Functions* (la dificultad computacional intrínseca de funciones) (1964). Cobham enfatizó el término "intrínseco", quien además estaba en otra teoría independiente a las máquinas. Lo cual condujo en 1965 a la identificación de la clase de problemas que se pueden resolver en tiempo acotado por un polinomio sobre la longitud de la entrada.

La diferencia entre *algoritmos de tiempo polinomial y algoritmos de tiempo exponencial* fue hecha por primera vez en 1953 por Von Neumann. La noción P para la clase de los problemas resolubles en tiempo polinomial fue introducida un poco después por Richard Karp (1972).

Se puede considerar que la teoría de la *NP-completitud* como el desarrollo más importante de la *complejidad algorítmica*. Esta clase *NP* consta de todos los problemas decidibles en tiempo polinomial por una máquina de Turing no determinista. Stephen Cook en 1971 introduce la noción de *problema NP-completo* y demuestra que el problema de la satisfacibilidad booleana es *NP-completo*.

Demostrar que un problema es *NP-completo* equivale a demostrar que no tiene una solución determinista en tiempo polinomial, salvo que todos los problemas de *NP* estén en P, cuestión que aún no está demostrada.

2.3. Máquinas secuenciales y autómatas finitos

La *teoría de autómatas*, que abarca también al estudio de las *máquinas secuenciales*, tienen su origen en el campo de la *ingeniería eléctrica*. El matemático norteamericano Claude Shanon estableció las bases para la aplicación de la lógica matemática a los circuitos combinatorios, luego David Huffman en 1954 aplicó estos conocimientos a los circuitos secuenciales, donde utiliza dos conceptos el *estado de un autómata* y *tabla de transición*. La formalización de la *teoría de las máquinas secuenciales* y de los autómatas finitos (1956) fue gracias a las ideas de Shanon.

Por otra parte, y desde un punto de vista totalmente diferente, el concepto de *autómata finito* aparece en 1943 en el artículo de Warren McCulloch y Walter Pitts titulado *A Logical Calculus of the Ideas Immanet in Nervous Activity*, donde proponen un modelo lógico (neurona artificial) con el propósito de simular la actividad de una neurona biológica. A consecuencia de esto, se han desarrollado asociaciones de neuronas para formar *redes neuronales artificiales* (RNA). De manera general, una RNA es una colección de neuronas, conectadas a otras neuronas o entradas externas, y con una salida que propaga las señales por múltiples caminos. Cada neurona (procesa-

dor) determina las entradas que recibe, luego las modifican para conseguir el objetivo previsto. Es lo que llamamos *función de aprendizaje*. Es decir, una RNA podía *aprender* de sus propios errores, esto a través de un proceso inductivo a partir de un conjunto de ejemplos. Las características que hacen interesantes a las RNAs son su capacidad de *aprender*, *memorizar*, *generalizar* o *abstraer*. Ahora bien las aportaciones de estas redes neuronales a la *teoría de la computación* son algunos paradigmas, tales como: el *cálculo paralelo*, el *aprendizaje inductivo* y su capacidad para realizar *cálculos aproximados* por medio de interpolación.

Posteriormente en 1951 Kleene realiza un informe donde demuestra la equivalencia entre los *conjuntos regulares*, los cuales pueden ser descritos mediante expresiones regulares y los *lenguajes reconocidos por un autómata finito*, este informe lo llamó "dos formas de definir una misma cosa".

Cuando un autómata se usa para modelar la construcción del hardware o software es muy importante examinar el problema, que consiste en encontrar el *autómata mínimo* equivalente a uno dado. A este problema se ocuparon J. Huffman y Eduard Moore, encontrando algoritmos para minimizar un autómata de estados finitos y demostraron que para un autómata de *n* estados estos requerían n^2 pasos. Mucho después, en 1971 Jhon E. Hopcroft encontró un método que lo hacía en $O(n \times log(n))$ pasos.

Finalmente podemos resumir los descubrimientos que se originaron y que se relacionan unos con otros, de la siguiente manera:

- Kleene en su intento de entender los trabajos de McCulloch y Pitts, abstrajo el concepto de *autómata finito* a partir de las redes de neuronas y el concepto de *expresión regular* a partir del cálculo lógico del modelo de McCulloch-Pitts.

- Myhill a partir de los conceptos de autómatas finitos de Kleene obtuvo el de *diagrama de transición* (deterministas) y a los eventos los redujo a la unión de clases de equivalencia.

Siguiendo esta lógica de trabajo, se ha construido y desarrollado una teoría abstracta de autómatas con una fuerte base matemática en las últimas décadas, y a todo esto Michael Arbib en 1969, lo llamó: "la matemática pura de la informática".

2.4. Gramáticas y lenguajes formales

Una gran distinción entre los *lenguajes formales* que tienen reglas sintácticas y semánticas rígidas, concretas y bien definidas contra los *lenguajes naturales* cuya sintaxis y semántica no se pueden controlar fácilmente y para caracterizar cada uno de los lenguajes es necesario un conjunto de reglas gramaticales adecuadas.

Noam Chomsky propone en 1956 tres modelos para describir los lenguajes, los cuales posteriormente formarían la base de la jerarquía de los tipos de lenguajes (1959), teniendo un gran impacto en el desarrollo de los lenguajes de programación. Chomsky hizo una clasificación de las gramáticas en base a sus producciones y distinguió cuatro clases fundamentales de lenguajes, además de la inclusión entre lenguajes.

La *teoría de los lenguajes formales* sin la intención de su creador resultó tener una relación muy fuerte y sorprendente con la *teoría de autómatas* y la *computabilidad*. Donde se hizo notar que los lenguajes propuestos por Chomsky tienen una equivalencia entre las máquinas abstractas, en donde a partir de restricciones de las gramáticas, le corresponde un tipo de máquina abstracta.

Cada uno de estos tipos de máquinas es capaz de resolver problemas más complejos desde los autómatas finitos (AF) hasta las máquinas de Turing (MT). Podemos notar que aunque las máquinas establecidas unos 20 años atrás resultaron reconocedoras de los lenguajes formales propuestos por Chomsky quien no se esperaba esta relación de equivalencia.

3. Lenguajes regulares

Los lenguajes regulares se llaman así porque sus palabras contienen "regularidades" o repeticiones de los mismos componentes, como por ejemplo en el Lenguaje L_1 siguiente:

$$L_1 = \{ab, abab, ababab, abababab, ...\}$$

En este ejemplo se aprecia que las palabras de L_1 son simplemente repeticiones de "ab" cualquier numero de veces. Aquí la "regularidad" consiste en que las palabras contienen "ab" algún número de veces.

Otro ejemplo más complicado sería el lenguaje L_2:

$$L_2 = \{abc, cc, abab, abccc, ababc, \ldots\}$$

La regularidad en L_2 consiste en que sus palabras empiezan con repeticiones de "ab" seguidas repeticiones de "c". Similarmente es posible definir muchos otros lenguajes basados en la idea de repetir esquemas simples. Esta es la idea básica para formar los *lenguajes regulares*.

Adicionalmente a las repeticiones de esquemas simples, vamos a considerar que los lenguajes finitos son también regulares por definición. Por ejemplo, el lenguaje $L_3 = \{anita, lava, la, tina\}$ es *regular*.

Finalmente, al combinar los lenguajes uniéndolos o concatenándolos, también se obtiene un lenguaje regular. Por ejemplo

$$L_1 \cup L_2 = \{anita, lava, la, tina, ab, abab, ababab, abababab, \ldots\}$$

es regular. También es regular una concatenación como:

$$L_3 L_3 = \{anitaanita, anitalava, anitala, anitatina, lavaanita, lavalava, lavala, lavatina, ...\}$$

3.1. Definición formal de lenguajes regulares

Definición 1 *Un lenguaje L es regular [3] si y sólo si se cumple al menos una de las condiciones siguientes:*

- L es finito;
- L es la unión o la concatenación de otros lenguajes regulares R_1 y R_2, $L = R_1 \cup R_2$ o $L = R_1 R_2$ respectivamente.
- L es la cerradura de Kleene de algún lenguaje regular, $L = R^*$.

Esta definición nos permite construir expresiones en la notación de conjuntos que representan lenguajes regulares.

Ejemplo.- Sea el lenguaje *L* de palabras formadas por a y b, pero que empiezan con a, como aab, ab, a, $abaa$, etc. Probar que éste lenguaje es regular, y dar una expresión de conjuntos que lo represente.

Solución.- El alfabeto es $\Sigma = \{a, b\}$. El lenguaje L puede ser visto como la concatenación de una a con cadenas cualesquiera de a y b; ahora bien, éstas últimas son los elementos de $\{a, b\}^*$, mientras que el lenguaje que sólo contiene la palabra a es $\{a\}$. Ambos lenguajes son regulares. Entonces su concatenación es $\{a\}\{a, b\}^*$, que también es regular.

3.2. Expresiones regulares

La noción de conjuntos nos permite describir los lenguajes regulares, pero es mejor una notación en donde las representaciones de los lenguajes sean simplemente texto (cadenas de caracteres). Así las representaciones de los lenguajes regulares serían simplemente palabras de un lenguaje. Con estas ideas vamos a definir un lenguaje, a través de las expresiones regulares, en donde cada palabra va a denotar un lenguaje regular.

Definición 2 *Sea Σ un alfabeto. El conjunto de las expresiones regulares (ER) [3] sobre Σ contienen las cadenas en el alfabeto $\Sigma \cup \{\text{“}\Lambda\text{”}, \text{“} + \text{”}, \text{“} \cdot \text{”}, \text{“} * \text{”}, \text{“(”}, \text{“)”}, \text{“}\Phi\text{”}\}$ que cumplen con lo siguiente:*

1. “Λ” y “Φ” $\in ER$.
2. $\sigma \in \Sigma$, entonces $\sigma \in ER$.
3. Si $E_1, E_2 \in ER$, entonces “(”E_1“$+$”E_2“)” $\in ER$, “(”E_1“$\cdot$”E_2“)” $\in ER$, “(”E_1“)$*$” $\in ER$.

Las ER son simplemente fórmulas cuyo propósito es representar cada una de ellas un lenguaje, en este caso un *lenguaje regular*. Así, el significado de una ER es simplemente el lenguaje que ella representa.

Para comprender intuitivamente la manera en que las ER representan lenguajes, consideremos el proceso de verificar si una palabra dada w pertenece o no al lenguaje representado por una ER dada. Vamos a decir que una palabra *“empata”* con una expresión regular si es parte del lenguaje que ésta representa.

La palabra vacía ε “empata” con la ER Λ.

Una palabra de una letra como “a” empata con una ER consistente en la misma letra “a”, “b” empata “b”, etc.

Luego, una palabra $w = uv$, esto es w está formada de dos pedazos u y v, empata con una expresión $(U \cdot V)$ a condición de que u empate con U y v empate con V. Un ejemplo más abc empata con $(a \cdot (b \cdot c))$ porque abc puede ser dividida en a y bc, y a empata con a en la ER, mientras que bc empata con $(b \cdot c)$ separando b y c de la misma manera.

Similarmente, cuando la ER es de la forma $(U + V)$, puede empatar con una palabra w cuando esta empata con U o bien con V. Por ejemplo, bc empata $(a + (b \cdot c))$.

Una palabra w empata con una expresión U^* cuando w puede ser partida en pedazos $w = w_1 w_2, ...$ de tal manera que cada pedazo w_i empata con U. Por ejemplo, $caba$ empata con $(((c + b) \cdot a))^*$ porque puede partirse en los pedazos ca y ba, y ambos empatan con $(((c + b) \cdot a))$, lo cual es fácil de verificar.

Con objeto de hacer la notación menos pesada, las ER se pueden simplificar de la siguiente manera:

- Omitiendo las comillas " ".
- Se eliminan los paréntesis innecesarios. Se supone una precedencia de operadores en el orden siguiente: primero "*", luego "$\cdot$" y finalmente "+". Además se supone que los operadores "$\cdot$" y "+" son asociativos.
- Eventualmente se omite el operador "$\cdot$", suponiendo que éste se encuentra implícito entre dos subexpresiones contiguas.

Ejemplo.-Encontrar una expresión regular para el lenguaje en $\{a, b\}^*$ en el que inmediatamente antes de toda b aparece una a.

Solución.-La expresión regular ER es: $(a + ab)^*$.

Algunos ejemplos de palabras generadas por esta expresión regular son las siguientes: a, aa, ab, $abab$, aab, $aaababab$, etc.

3.3. Gramáticas regulares

Además de las expresiones regulares existe otra forma de representar los lenguajes regulares, y a esta forma se les llama *gramáticas regulares*.

Para ello primeramente vamos a ver que son las gramáticas y como nos ayudan a representar a los lenguajes formales.

3.4. Gramáticas formales

La representación de los lenguajes regulares se fundamentan en la noción de *gramática formal*. Intuitivamente, una *gramática* es un conjunto de reglas para formar correctamente las frases de un lenguaje. La formalización que se presenta de la noción de gramática es debida a Chomsky, y está basada en las llamadas *reglas gramaticales*.

Una *regla* es una expresión de la forma $\alpha \rightarrow \beta$, en donde tanto α como β son cadenas de símbolos en donde pueden aparecer tanto elementos del alfabeto Σ como unos nuevos símbolos, llamados *variables* (no terminales). Los símbolos que no son variables son *constantes* (terminales). Por ejemplo, una posible regla gramatical es $X \rightarrow aX$. La aplicación de una regla $\alpha \rightarrow \beta$ a una palabra $u\alpha v$ produce la palabra $u\beta v$. En consecuencia, las reglas de una gramática pueden ser vistas como reglas de reemplazo. Por ejemplo, si tenemos una cadena de símbolos $bbXa$, le podemos aplicar la regla $X \rightarrow aX$, dando como resultado la nueva cadena $bbaXa$.

3.5. Definición formal de gramática regular

Ahora nos centraremos en las gramáticas cuyas reglas son de la forma $A \to \alpha B$ o bien $A \to \alpha$, donde A y B son variables, y a es un carácter terminal. A estas gramáticas se les llama *regulares*.

La idea de aplicar una gramática es que se parte de una variable, llamada *símbolo inicial*, y se aplican repetidamente las reglas gramaticales, hasta que ya no haya variables en la palabra. En ese momento se dice que la palabra resultante es *generada* por la gramática, o en forma equivalente, que la palabra resultante es parte del lenguaje de esa gramática.

Ahora damos la definición de una gramática regular formalmente:

Definición 3 *Una gramática regular es una cuádrupla* (V, Σ, R, S) *[3] en donde:*

- V es un alfabeto de *variables* (no terminal)
- Σ es un alfabeto de *constantes* (terminal)
- R conjunto de reglas, es un subconjunto finito de $V \times (\Sigma V \cup \Sigma)$.
- S símbolo inicial, es un elemento de V.

Ejemplo.-Sea una gramática

$$(\{S, A, B\}, \{a, b\}, \{(S, aA), (S, bA), (A, aB), (A, bB), (A, a), (B, aA), (B, bA)\}, S)$$

con ésta gramática tenemos las siguientes reglas:

1. $S \to aA$
2. $S \to bA$
3. $A \to aB$
4. $A \to bB$
5. $A \to a$
6. $B \to aA$
7. $B \to bA$

Usualmente las reglas no se escriben como pares ordenados (A, aB), como lo requeriría la definición anterior, sino como $A \to aB$; esto es simplemente cuestión de facilidad de notación.

La aplicación de una gramática se formaliza con las siguientes nociones:

1. La cadena uXv deriva en un paso una cadena $u\alpha v$, escrito como $uXv \Rightarrow u\alpha v$, si hay una regla $X \to \alpha \in R$ en la gramática.
2. Una cadena $w \in \Sigma^*$ (esto es, formada exclusivamente por constantes) es derivable a partir de una gramática G si existe una secuencia de pasos de derivación $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w$.

Definición 4 *El lenguaje generado por una gramática* G*,* $L(G)$*, es igual al conjunto de las palabras derivables a partir de un símbolo inicial.*

Esto es, $L(G) = \{w \in \Sigma^* | S \stackrel{*}{\Rightarrow} w\}$

Ejemplo.-Proponer una gramática que genere el lenguaje de las palabras en $\{a, b\}$ que contienen la subcadena bb, como abb, $ababba$, etc.

Vamos a utilizar las variables como memorias para "recordar" situaciones. Así tendremos las siguientes variables:

- A, que recuerda que aún no se produce ninguna b.
- B, que recuerda que se produjo una b.
- C, que recuerda que ya se produjeron las dos $b's$.

Ahora podemos proponer reglas, preguntándonos a qué situación se llega al producir una a o b. Por ejemplo, a partir de A, si se produce una a se debe llegar a la misma A, pero si llega una b se llegará a la variable B. Con estas ideas se propone la siguiente gramática:

$$(\{A, B, C\}, \{a, b, c\}, \{(A, aA), (A, bB), (B, aA), (B, bC), (C, aC), (C, bC)\}, \{A\})$$

de donde tenemos las siguientes reglas:

1. $A \to aA$
2. $A \to bB$
3. $B \to aA$
4. $B \to bC$
5. $C \to aC$
6. $C \to bC$

Finalmente, para terminar la producción de una palabra hecha solamente de constantes es necesaria al menos una regla que no produzca variables en su lado derecho. Tal regla no se encuentra aún en la gramática dada. Como las palabras correctas tienen bb, pensamos que una regla adicional podría ser $C \to a$ y también $C \to b$. En efecto, con tales reglas podemos producir por ejemplo, la palabra $abba$, mediante la derivación siguiente:

$$A \Rightarrow aA \to abB \Rightarrow abbC \Rightarrow abba$$

Sin embargo, también podemos verificar que la palabra abb, que pertenece al lenguaje, no puede producirse con las reglas dadas. Hace falta aún otra regla, $B \to b$, con ésta última regla se completa nuestra gramática. Su representación formal es:

$$(\{A, B, C\}, \{a, b, c\}, \{(A, aA), (A, bB), (B, aA), (B, bC), (C, aC), (C, bC)(C, a), (C, b), (B, b)\}, \{A\})$$

4. Lenguajes libres de contexto

Los *lenguajes libres de contexto* (abreviado LLC) forman una clase de lenguajes más amplia que los *lenguajes regulares*, de acuerdo con la jerarquía de Chomsky. Estos lenguajes son importantes tanto desde el punto de vista teórico, por relacionar las llamadas *gramáticas libres de contexto* con los autómatas de pila[2], como desde el punto de vista práctico, ya que casi todos los lenguajes de programación están basados en los LLC. Para poder entender este tipo de lenguajes es necesario entender algunos conceptos que se mencionan a continuación.

- Regla.- es una expresión de la forma $\alpha \rightarrow \beta$, en donde tanto α como β son cadenas de símbolos en donde se pueden aparecer tanto elementos del alfabeto Σ (llamados constantes) como unos nuevos símbolos, llamados variables.
- Gramática.- es básicamente un conjunto de reglas.

Consideremos, por ejemplo, la siguiente gramática para producir un pequeño subconjunto del idioma español:

1. $< frase > \rightarrow < sujeto >< predicado >$
2. $< sujeto > \rightarrow < articulo >< sustantivo >$
3. $< articulo > \rightarrow el|la$
4. $< sustantivo > \rightarrow perro|luna$
5. $< predicado > \rightarrow < verbo >$
6. $< verbo > \rightarrow brilla|corte$

donde el símbolo "|" separa varias alternativas. En ésta gramática se supone que las variables (no terminales) son: $< frase >$, $< sujeto >$, $< articulo >$, $< sustantivo >$, $< predicado >$ y $< verbo >$, mientras que las constantes (terminales) son: el, la, $perro$ y $luna$. La variable $< frase >$ será considerada el símbolo inicial.

4.1. Gramáticas libres de contexto

Podemos ver que la gramática del español dada arriba es una (gramática libre de contexto) GLC, pero no podría ser una gramática regular, pues hay varias reglas que no corresponden al formato de las reglas de las gramáticas regulares. Se ve por lo tanto, que el formato de las reglas es menos rígido en las GLC que en las gramáticas regulares, y así toda gramática regular es GLC pero no viceversa.

Por ejemplo, el lenguaje $\{a^n b^n\}$ que no es regular, tiene la gramática libre de contexto con las siguientes reglas:

1. $S \rightarrow aSb$
2. $S \rightarrow ab$

[2]Para saber acerca del estudio de los autómatas de pila, ver [2]

Como vimos en el caso de las gramáticas regulares, aplicar una regla $X \to \alpha$ de una gramática consiste en reemplazar X por α en una palabra. Por ejemplo, la regla $S \to aSb$ se puede aplicar a una palabra $aaSbb$ para obtener la palabra $aaaSbbb$, en donde es fácil ver que reemplazamos S por aSb.

El proceso de aplicar una regla se le conoce como "paso de derivación", y se denota, usando una flecha gruesa "⇒", como en $aaSbb \Rightarrow aaaSbbb$ (aplicando una regla $S \to aSb$). Una secuencia de pasos de derivación a partir de una variable especial de la gramática llamada "símbolo inicial" se llama simplemente *derivación*. Por ejemplo, una derivación de la palabra "$aaabbb$" utilizando la gramática de $\{a^n b^n\}$ sería (suponiendo que S es el símbolo inicial):

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

Como un ejemplo adicional, la gramática con las reglas siguientes permite generar expresiones aritméticas con sumas y multiplaciones de enteros:

1. $E \to E + T$
2. $E \to E$
3. $T \to T * F$
4. $T \to F$
5. $F \to CF$
6. $F \to C$
7. $C \to 0|1|2|3|4|5|6|7|8|9$

El símbolo inicial aquí es E, las constantes son $+$, $*$ y las cifras $0..,9$; E, T, F, C son variables.

4.2. Definición formal de gramática libre de contexto

Definición 5 *Una gramática libre de contexto es una cuádrupla* (V, Σ, R, S) *[3] en donde:*

- V es un *alfabeto de variables*, también llamadas *no-terminales*.
- Σ es un *alfabeto de constantes*, también llamadas *terminales*. Suponemos que V y Σ son disjuntos, esto es, $V \cap \Sigma = \emptyset$.
- R, el conjunto de *reglas*, es un subconjunto finito de $V \times (V \cup \Sigma)$.
- S, el *símbolo inicial*, es un elemento de V.

Ejemplo.- la gramática de $\{a^n b^n\}$ que presentamos con anterioridad se representa formalmente como:

$$(\{S\}, \{a, b\}, \{(S, aSb), (S, ab)\}, S)$$

Usualmente las reglas no se escriben como pares ordenados (X, α), sino como $X \to \alpha$; esto es simplemente cuestión de notación.

Definición 6 *Una cadena $w \in (V \cup \Sigma)^*$ es derivable a partir de una gramática (V, Σ, R, S) si hay al menos una secuencia de pasos de derivación que la produce a partir del símbolo inicial S, esto es:*

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow w.$$

Definición 7 *El lenguaje $L(G)$ generado por una gramática (V, Σ, R, S) es el conjunto de palabras hechas exclusivamente de constantes, que son derivables a partir del símbolo inicial.*

$$L = \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$$

Las reglas permiten establecer una relación entre cadenas en $(V \cup \Sigma)^*$, que es la *relación de derivación*, $\Rightarrow_G$ para una gramática G. Esta relación se define de la siguiente manera:

Definición 8 *$\alpha \Rightarrow_G \beta$ si y sólo si existen cadenas $x, y \in (V \cup \Sigma)^*$, tales que $\alpha = xuy$, $\beta = xvy$, y existe una regla $u \to v$ en R.*

La cerradura reflexiva y transitiva de $\Rightarrow_G$ se denota por $\Rightarrow^*_G$. Una palabra $w \in \Sigma^*$ es *derivable* a partir de G si existe una secuencia de derivación $S \Rightarrow^*_G w$.

Definición 9 *El lenguaje generado por una gramática G, esto es, $L(G)$, es igual a $\{w \in \Sigma^* | S \Rightarrow^*_G w\}$.*

4.3. Definición formal de lenguajes libres de contexto

Para definir este tipo de lenguaje de manera formal generado por una gramática $G = (V, T, P, S)$. Debemos desarrollar la notación para representar una derivación.

Primero definimos dos relaciones $\underset{G}{\Rightarrow}$ y $\overset{*}{\underset{G}{\Rightarrow}}$ entre cadenas de $(V \cup T)^*$. Si $A \to \beta$ es una producción de P y α y γ son cualesquiera cadenas de $(V \cup T)^*$, entonces $\alpha A \gamma \underset{G}{\Rightarrow} \alpha\beta\gamma$. Decimos que la producción $A \to \beta$ está aplicada a la cadena $\alpha A \gamma$ para obtener $\alpha B \gamma$ o que $\alpha A \gamma$ deriva directamente a $\alpha B \gamma$ en la gramática G. Dos cadenas están relacionadas por $\underset{G}{\Rightarrow}$ exactamente cuando la segunda se obtiene a partir de la primera mediante una aplicación de una producción.

Supóngase que α_1, α_2,..., α_m son cadenas de $(V \cup T)^*$, $m \geq 1$ y $\alpha_1 \underset{G}{\Rightarrow} \alpha_2$, $\alpha_2 \underset{G}{\Rightarrow} \alpha_3$, ..., $\alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$.

Entonces decimos que $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$ o α_1 *deriva* a α_m en la gramática *G*. Es decir, $\overset{*}{\underset{G}{\Rightarrow}}$ es la cerradura transitiva y reflexiva de $\underset{G}{\Rightarrow}$. Alternativamente, $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ se concluye de α mediante la aplicación de cero o más producciones de *P*. Podemos notar que $\alpha \underset{G}{\Rightarrow} \alpha$ para la cadena α. Por lo general, si está claro que gramática *G* está involucrada, utilizamos $\Rightarrow$ en lugar de $\underset{G}{\Rightarrow}$, y $\overset{*}{\Rightarrow}$ en lugar de $\overset{*}{\underset{G}{\Rightarrow}}$. Si α deriva a β exactamente en *i* pasos, decimos que $\alpha \Rightarrow \beta$.

El *lenguaje generado* [2] por G denotado por $L(G)$ es $\{w | w$ está en T^* y $S \overset{*}{\underset{G}{\Rightarrow}} w\}$. Esto es, una cadena está en *L(G)* si:

- La cadena consiste solamente en terminales.
- La cadena puede derivarse a partir de S.

Llamamos a L un *lenguaje libre de contexto* (LLC) si éste es $L(G)$ para alguna GLC G. Una cadena de terminales y variables α se conoce como *forma oracional* si $S \overset{*}{\Rightarrow} \alpha$. Definimos las gramáticas G_1 y G_2 como *equivalentes* si $L(G_1) = L(G_2)$.

5. Lenguajes sensibles del contexto

Este tipo de lenguajes son definidos por las *gramáticas sensibles del contexto*, dichas gramáticas definen esta clase intermedia de lenguajes, que se sitúan entre entre los lenguajes libres de contexto y los lenguajes recursivos.

Casi cualquier lenguaje que uno pueda concebir es sensible al contexto, a excepción de los *lenguajes recursivos*

Las reglas son de la forma $\alpha \rightarrow \beta$, donde α y β no permiten ε de una producción, i.e., no se permite la palabra vacía tanto para el lado izquierdo como para el lado derecho. Sin embargo, pueden contener cualquier cantidad de variables (no terminales) y constantes (terminales).

Las *gramáticas sensitivas del contexto* son estrictamente más poderosas que las *gramáticas libres del contexto*; un ejemplo es el lenguaje de las cadenas de la forma $\{a^n b^n c^n\}$, para el que no hay ninguna gramática libre de contexto (GLC).

5.1. Definición formal de gramáticas sensibles del contexto

Definición 10 *Una gramática sensible del contexto es una cuádrupla $G = (N, \Sigma, S, P)$ [2] en donde:*

- N es un alfabeto de símbolo no terminales (variables)
- Σ es un alfabeto de símbolos terminales con $N \cup \Sigma = \emptyset$
- $S \in N$ es el símbolo inicial
- P es el conjunto finito de producciones de la forma $\alpha \rightarrow \beta$, donde α y $\beta \in (N \cup \Sigma)^+$ y $|\alpha| \leq |\beta|$.

Como ejemplo, dado el lenguaje $\{a^n b^n c^n | n \geq 1\}$, la gramática que genera dicho lenguajes es la siguiente:

$$G = (\{S, A, B\}, \{a, b, c\}, \{S\}, \{(S, abc|aAbc), (Ab, bA), (Ac, Bbcc), (bB, Bb), (aB, aa|aaA)\})$$

de donde obtenemos las producciones de esta gramática, las cuales son:

1. $S \rightarrow abc|aAbc$
2. $Ab \rightarrow bA$
3. $Ac \rightarrow Bbcc$
4. $bB \rightarrow Bb$
5. $aB \rightarrow aa|aaA$

con lo que tenemos un ejemplo de un *lenguaje sensible del contexto* que no es libre del contexto. Para verificar que la gramática representa al lenguaje, vamos a generar la palabra $aaabbbccc$ la cual se puede hacer de la siguiente forma:

$$S \Rightarrow_1 aAbc \Rightarrow_2 abAc \Rightarrow_3 abBbcc \Rightarrow_4 aBbbcc \Rightarrow_5 aaAbbcc \Rightarrow_2 aabAbcc \Rightarrow_2 aabbAcc \Rightarrow_3 aabbBbccc \Rightarrow_4 aabBbbccc \Rightarrow_4 aaBbbbccc \Rightarrow_5 aaabbbccc$$

Para ello realizamos 11 derivaciones o pasos para generar la palabra $aaabbbccc$.

Analicemos la siguiente gramática:

$G = (\{S, A, B, T, X, Y\}, \{a, b, c\}, \{S\}, \{(S, aBTc), (T, ABTc), (T, ABc), (BA, Bx), (BX, YZ), (YX, AX$

que nos genera el lenguaje ($L = \{a^n b^n c^n | n \geq 1\}$), de donde obtenemos las siguientes producciones:

1. $S \rightarrow aBTc$
2. $T \rightarrow ABTc$
3. $T \rightarrow ABc$
4. $BA \rightarrow Bx$
5. $BX \rightarrow YX$
6. $YX \rightarrow AX$
7. $AX \rightarrow AB$
8. $aA \rightarrow aa$
9. $aB \rightarrow ab$
10. $bB \rightarrow bb$

Podemos observar con un poco más de detalle que, las reglas 1 a 3 generan *A*, *a*, *B* y *c* no necesariamente en orden (las *A* y *B* van a estar alternadas). Luego las reglas 4 a 7 permiten reordenar las *A* y *B*, para que todas las *A* queden antes que todas las *B*, y finalmente las reglas 8 a 10 permiten generar los terminales solamente cuando las letras están en el orden correcto. Como un ejemplo, la palabra *aaabbbccc* se puede generar de la forma siguiente:

$S \Rightarrow_1 aBTc \Rightarrow_2 aBABTcc \Rightarrow_3 aBABABccc \Rightarrow_4 aBXBXBccc \Rightarrow_5 aYXYXBccc \Rightarrow_6 aAXAXBccc \Rightarrow_7 aABABBccc \Rightarrow_4 aABXBBccc \Rightarrow_5 aAYXBBccc \Rightarrow_6 aAAXBBccc \Rightarrow_7 aAABBBccc \Rightarrow_8 aaABBBccc \Rightarrow_9 aaabBBccc \Rightarrow_{10} aaabbBccc \Rightarrow_{10} aaabbbccc$.

Como podemos notar en esta gramática y la anterior nos generan el mismo lenguaje: ($L = \{a^n b^n c^n | n \geq 1\}$), aunque cada gramática tiene un número diferente de reglas de producción, con esto podemos deducir el siguiente lema:

- Si dos o más gramáticas generan el mismo lenguaje, entonces las gramática son equivalentes.

Como última observación vemos que en la última gramática se hicieron 15 derivaciones (pasos), mientras que en la primera 11 derivaciones, por lo cual tenemos que a mayor número de reglas de producción mayor es número de pasos de derivación, que obviamente se ve reflejado en términos de costo de computación.

Como toda *gramática libre del contexto* se puede poner en la forma $A \rightarrow \alpha$ o también $A \Rightarrow BC$. Puesto que las producciones de esta forma satisfacen la definición de *gramáticas sensibles del contexto*. Por tanto podemos deducir lo siguiente:

- El conjunto de los lenguajes sensibles del contexto contiene el conjunto de los lenguajes libres del contexto.

6. Lenguajes recursivos y recursivamente enumerables

Estos lenguajes también se les conoce como lenguajes estructurados por frase, y son los lenguajes tipo 0 dentro de la jerarquía de Chomsky. Un lenguaje *L* sobre un alfabeto Σ se dice que es *recursivamente enumerable* si es aceptado por una máquina de Turing[3].

Las gramáticas que describe a este tipo de lenguajes se les conoce como *gramáticas no restringidas* o bien *gramáticas estructuradas por frases*. Las gramáticas no restringidas a las que hacemos alusión en esta sección, se conocen así por que no tiene restricción alguna en cuanto a sus reglas de producción. Así que, tanto el lado izquierdo como el derecho de las reglas de producción pueden contener cualquier cadena finita de terminales (constantes) y no terminales (variables), siempre y cuando exista por lo menos un no terminal en el lado izquierdo.

Este tipo de gramáticas generan los lenguajes conocidos como *lenguajes estructurados por frases* los cuales se subdividen en *lenguajes recursivos y recursivamente enumerables*.

Las producciones que generan estas gramáticas son de la forma $\alpha \rightarrow \beta$, en donde α y β son cadenas arbitrarias de símbolos (terminales y no terminales), y además $\alpha \neq \varepsilon$.

6.1. Definición formal de gramática no restringida

Definición 11 *Una gramática no restringida es una cuádrupla* $G = (N, \Sigma, S, P)$ *[2] en donde:*

- N es un alfabeto de símbolos no terminales.
- Σ es un alfabeto de símbolos terminales con $N \cap \Sigma = \emptyset$.
- $S \in N$ es el símbolo inicial.
- P es un conjunto finito de producciones de la forma $\alpha \rightarrow \beta$, donde $\alpha \in (N \cup \Sigma)^+$ y $\beta \in (N \cup \Sigma)^*$ (i.e., $P \subset (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ y es un conjunto finito).

Como ejemplo, consideremos la siguiente *gramática no restringida* que genera el lenguaje $L = \{a^{2k} | k > 0\}$ cuya representación formal es:

$G = (\{S, A, B, C, D, E\}, \{a, \varepsilon\}, \{S\}, \{(S, ACaB), (Ca, aaC), (CB, DB|E), (aD, Da), (AD, AC), (aE, Ea), (AE, \varepsilon)\})$

de donde obtenemos las siguientes producciones:

1. $S \rightarrow ACaB$
2. $Ca \rightarrow aaC$
3. $CB \rightarrow DB|E$
4. $aD \rightarrow Da$
5. $AD \rightarrow AC$
6. $aE \rightarrow Ea$

[3]Para conocer acerca del estudio de las máquinas de Turing, ver [1] y [3]

7. $AE \rightarrow \varepsilon$

Para observar más a detalle de cómo se generan cadenas en *L*, consideremos la derivación $a^{2^2} = a^4$. Tendremos:

S ⇒ *ACaB* ⇒ *AaaCB* ⇒ *AaaDB* ⇒ *AaDaB* ⇒ *ADaaB* ⇒ *ACaaB* ⇒ *AaaCaB* ⇒ *AaaaaCB* ⇒ *AaaaaE* ⇒ *AaaaEa* ⇒ *AaaEaa* ⇒ *AaEaaa* ⇒ *AEaaa* ⇒ *aaaa*

A y *B* actúan como marcadores de final de la cadena de *a's* que están siendo generadas. *C* se desplaza hacia la derecha al duplicarse el número de *a's* hasta que está junto a *B*, entonces se transforma en una *D*, *D* se desplaza hacia la derecha hasta que encuentra una *A*, y entonces se convierte en *C*. Cuando *CB* es reemplazada por *E*, termina la generación de *a's*. Entonces la *E* se desplaza hacia la izquierda hasta que encuentra la *A*, momento en el cual se elimina *AE*.

Se propone un ejemplo más de gramática no restringida que genera a $L = \{a^n b^n c^n | n \geq 1\}$, cuya representación formal de dicha gramática es la siguiente:

$G = (\{S, B, C\}, \{a, b, c\}, \{S\}, \{(S, aSBC|aBC), (CB, BC), (aB, ab), (bC, bc), (cC, cc)\})$

1. $S \rightarrow aSBC|aBC$
2. $CB \rightarrow BC$
3. $aB \rightarrow ab$
4. $bC \rightarrow bc$
5. $cC \rightarrow cc$

En la cadena que resulta sustituir el símbolo inicial, siempre hay el mismo número de *a's*, *B's* y *C's*. Es más, todas las *a's* preceden a las *B's* y a las *C's*. La producción *CB*→*BC* nos permiten intercambiar los no terminales *C* y *B*, obteniendo una cadena de la forma $a^n B^n C^n$. Las producciones *aB*→*ab* y *bB*→*bb*, transforman toda *B* en *b*, obteniéndose $a^n b^n C^n$. Finalmente, las producciones *bC*→*bc* y *cC*→*cc*, transforman la cadena en $a^n b^n c^n$.

Por otro lado, si *n* = 1, esta gramática puede derivar *abc* por medio de *S* ⇒ *aBC* ⇒ *abC* ⇒ *abc*.

Si $n \geq 1$, entonces la cadena $a^n b^n c^n$ se deriva por medio de:

$S \Rightarrow^{n-1} a^{n-1} S(BC)^{n-1} \Rightarrow a^n (BC)^n \Rightarrow^n a^n B^n C^n \Rightarrow^n a^n b^n C^n \Rightarrow^n a^n b^n c^n$

Por tanto, en *G* se deriva cualquier cadena de la forma $a^n b^n c^n$ para $n \geq 1$. De esto se deduce que $L(G) = \{a^n b^n c^n | n \geq 1\}$.

Analizando esta última gramática:

$$G = (\{S, B, C\}, \{a, b, c\}, \{S\}, \{(S, aSBC|aBC), (CB, BC), (aB, ab), (bC, bc), (cC, cc)\})$$

que es una gramática no restringida, podemos observar que se trata de una gramática sensible del contexto, como se analizó en la sección anterior (gramáticas sensibles del contexto) por lo que podemos deducir que las *gramáticas sensibles del contexto* están dentro de la *gramáticas no restringidas*, i.e., las gramáticas no restringidas contiene a las sensibles del contexto y por consecuencia, las no restringidas son más "poderosas" que las sensibles del contexto, las libres del contexto y las gramáticas regulares, por lo que podemos formular el siguiente teorema:

Teorema.- (la jerarquía de Chomsky) [6] Se tienen la siguientes afirmaciones:

1. La familia de los lenguajes regulares está estrictamente contenida en la familia de los lenguajes independientes del contexto.

2. La familia de los lenguajes independientes del contexto (que no contienen la palabra vacía) esta estrictamente contenida en la familia de los lenguajes sensibles al contexto.

3. La familia de los lenguajes sensibles al contexto esta estrictamente contenida en la familia de los lenguajes recursivamente enumerables.

7. Computación no convencional

La computación no-convencional es un conjunto de áreas de la ciencia que buscan la implementación de computación basado en el hecho de que la computación es un proceso por el cual los datos son manipulados: adquiriendolos (entrada), transformándolos (haciendo cálculos) y transfiríendolos (salida), i.e., cualquier forma de procesamiento ocurrida, ya sea de manera natural o manipulada de por algún medio no linael. La idea básica es:

1. Medir una entidad física.

2. Ejecutar una operación aritmética o lógica.

3. Darle valor a una cantidad física.

La computación no convencional se define de la siguiente manera: Un esquema de computación que actualmente es visto como no-convencional debe ser porque su desarrollo aún no es disponible o terminado [7].

La computación no-convencional para sus desarrollo, simulación, experimentación se auxilia de:

- Computación cuántica
- Computación química
- Computación biológica
- Computación óptica
- Computación natural
- Computación bio-inspirada
- Algoritmos genéticos
- Autómatas celulares

7.1. Autómata celular

Para poder predecir, interpretar e investigar sobre algunos sucesos en cuyos sistemas tiene una alto grado de complejidad, una de las formas más efectivas para su interpretación es simulación. Sin embargo, dicha complejidad es tal que no es suficiente para representar todos los fenómenos utilizando la matemática clásica. Por lo tanto, la simulación es una de las herramientas más precisas y utilizadas por los científicos o personas que se dedican al estudio de este tipo de sistema con una cierta complejidad.

Los autómatas celulares (AC) precisamente nos permiten este tipo de simulación para algunos sistemas complejos. Esta compuesto por elementos muy simples: una cuadricula con cuadros, que adoptan distintos valores y pasos discretos en tiempo determinado. En cada paso de tiempo el autómata celular evoluciona en base a reglas simples. El nuevo valor de cada celda (cuadro de la cuadrícula) se calcula en base a la actual y sus vecinos (celdas tanto de lado izquierdo como derecho). Esto se hace para cada una de las celdas de manera simultanea. A pesar de su simpleza, lo que realmente determina que sea un modelo complejo es que elementos simples originan situaciones y cálculos muy complejos.

7.2. Definición formal de Autómata Celular

Un autómata celular es una cuádrupla $A = (\Sigma, \mu, \varphi, c_0)$ [8] en donde:

- Σ es el conjunto finito de estados.
- μ es una conección local tal que $\mu = \{X_i, j, ..., n : d\}$.
- φ es la función de transición $\varphi : \mu \to \Sigma$.
- c_0 es la condición inicial del sistema.

Para comprender mejor esta definición, consideremos algunos de los conceptos que están inmiscuidos en la definición formal del autómata celular, los cuales:

- **Conjuntos de entes:** Se necesita saber cuántos objetos elementales van a formar la población del sistema. En principio no hay restricción a su número, pudiendo ser desde unos pocos hasta una infinidad. En ocasiones es importante situarlos sobre una región geográfica, identificándose entonces los entes con sus respectivas coordenadas geográficas.
- **Vencidades:** Para cada elemento del sistema es necesario establecer su vecindad, i.e., aquellos otros elementos que serán considerados como sus vecinos. En caso de asociar objetos con coordenadas de un sistema de referencia, el criterio suele ser construir la vecindad de un elemento dado con todos aquellos otros elementos que se encuentran a menos de una cierta distancia a radio, de forma que los más alejados no ejerzan influencia directa sobre él.
- **Conjunto de estados:** En cada instante, cada elemento deberá encontrarse en un cierto estado. El caso más sencillo corresponde a los elementos biestables, los cuales se pueden encontrar en sólo uno de dos estados posibles, 0 y 1, por ejemplo. Pero también el estado puede venir representado por un vector de componentes reales o por una cadena de un lenguaje formal.

- **Regla de transición local:** La regla de transición define la dinámica del sistema. Dado un elemento y un instante determinados, la regla devuelve el siguiente estado del elemento, para ello necesita como argumentos los estados actuales, tanto del elemento considerado como aquellos que conforman su vecindad. Las reglas de transición pueden ser deterministas o probabilistas, además, no todos los elementos necesitan obedecer a la misma regla.

7.3. Tipos de vecindad

La regla de evolución puede encontrar algunas variantes a través de la historia. Las mas comunes son:

1. **Vecindad de Von Neumann:** Definida para una célula central y sus vecinos ortogonales.
2. **Vecindad de Moore:** Además de los vecinos ortogonales se consideran los vecinos diagonales.
3. **Vecindad hexagonal:** La célula central es de forma hexagonal y cada lado corresponde a un vecino.

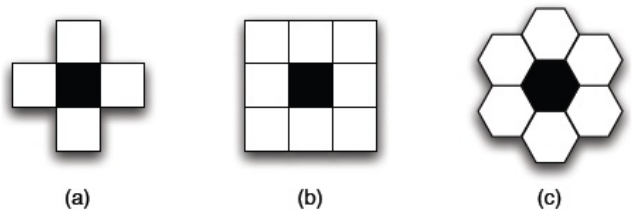

Figura 1: Juárez Martínez Genaro, *Tipos de vecindad en dos dimensiones*. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/

7.4. Clasificación de Wolfram

Wolfram a clasificado a los AC en una dimensión en cuatro grupos, los cuales son:

1. Un AC es de clase I si existe un estado estable $x_i \in \Sigma$ tal que todas las configuraciones finitas x_i evolucionan a la configuración homogénea de $x - i$.
2. Un AC es de clase II si existe un estado estable $x_i \in \Sigma$ tal que cualquier configuración finita tiene una evolución que llega a ser periódica.
3. un AC es de clase III si existe un estado estable $x_i \in \Sigma$ tal que para alguna pareja de configuraciones finitas x_i de c_i y c_j, es decidible si c_i evoluciona a c_j.
4. Un AC es de clase IV si incluye todo AC, i.e., esta clase contiene a las otras tres clases.

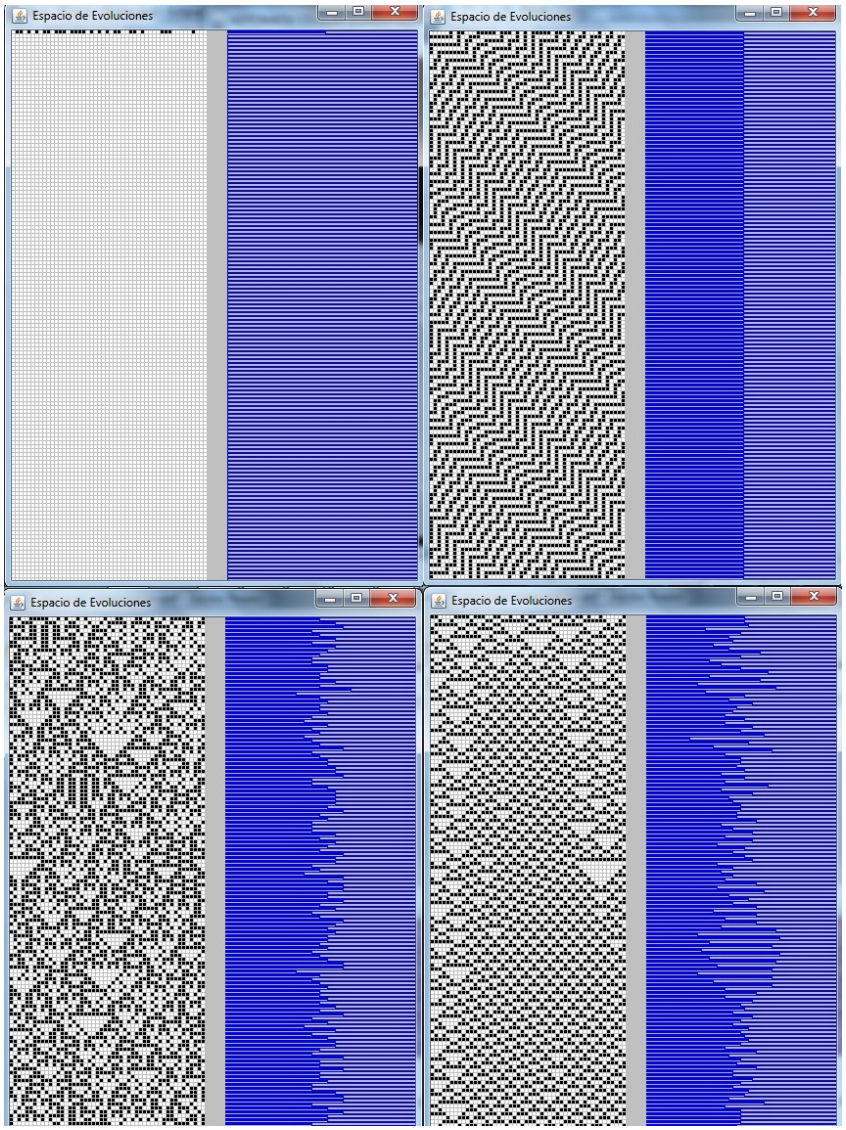

Figura 2: Regla 0, 15, 90 y 54; las 4 clases de un AC elemental de una dimensión.

7.5. Compuerta majority

En una compuerta *majority* en un quantum-dot basado en AC, cada célula cuántica cambia de polarización (estado) al recibir un estímulo. La célula cuántica-dot, puede tener dos estados (polarizaciones) -1 0 +1, como se ilustra en la siguiente figura. Luego tenemos la implementación e interpretación de la compuerta en un AC, como se observa en la siguiente figura.

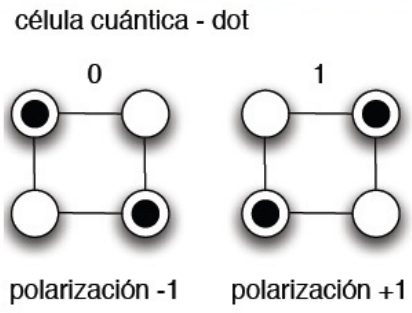

Figura 3: Juárez Martínez Genaro, *Polarización de una compuerta majority*. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/

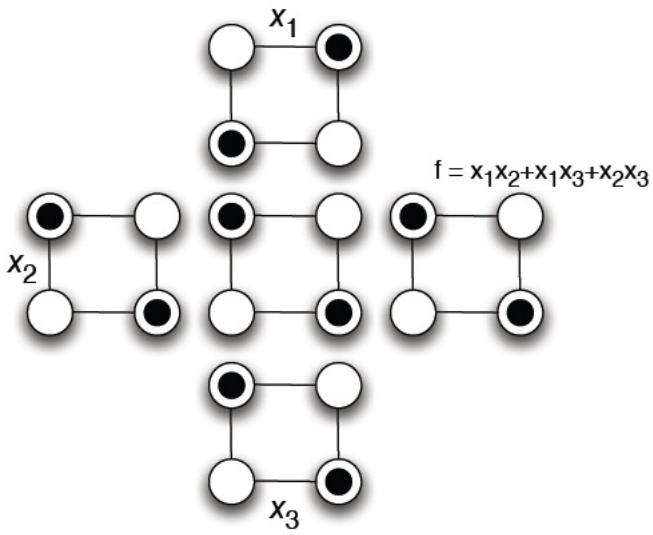

Figura 4: Juárez Martínez Genaro, *Implementación de una compuerta majority en un quantum-dot AC*. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/

8. Propagación de ondas en patrones, caso de estudio

El caso de estudio que se presenta, constituyen avances en un AC simulando compuertas lógicas a través de un sistema de competición en canales de comunicación, choques y propagación de ondas, dicho modelo implementa una compuerta *majority* en un AC bidimensional, binario y con función semi-totalística.

8.1. Computación en las reglas 2c22

En las reglas de autómatas 2c22 se seleccionó la regla 2622 para hacer las construcciones (patrones). Podemos hacer barreras fijas impenetrables auto-localizadas, llamadas localizaciones con configuraciones de *life*. Primitivamente, cada localización fija es una intersección de cuatro segmentos de 1 estado, cada segmento ocupa seis celdas en estado 1; los segmentos se interceptan

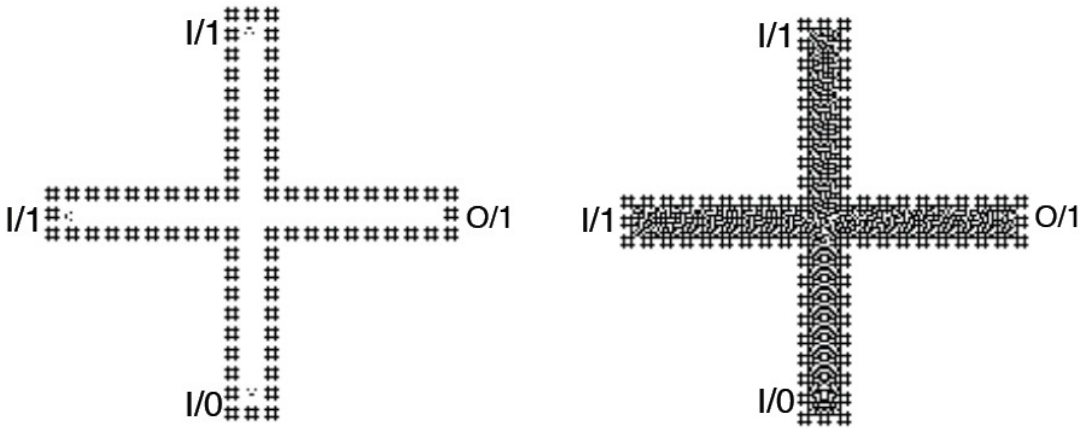

Figura 5: Juárez Martínez Genaro, *Implementación de la compuerta majority en un AC bidimensional, binario y con función semi-totalística*. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/

y cruzan para formar $2 * 2$ del dominio de las celdas en estado 0. Eventualmente, hay familias de patrones manipulando el ancho del canal y las posiciones de móvil auto-localizadas. Sin embargo, se ha utilizado una distancia mínima de móvil y fijo auto-localizadas, que se calcula como:

Intervalo = volumen de la naturaleza muerta - volumen del glider

El mínimo espacio entre las paredes limita el número de patrones en éstas. Como podemos apreciar en la siguiente figura. Hay dos tipos de patrones de propagación en los canales de in-

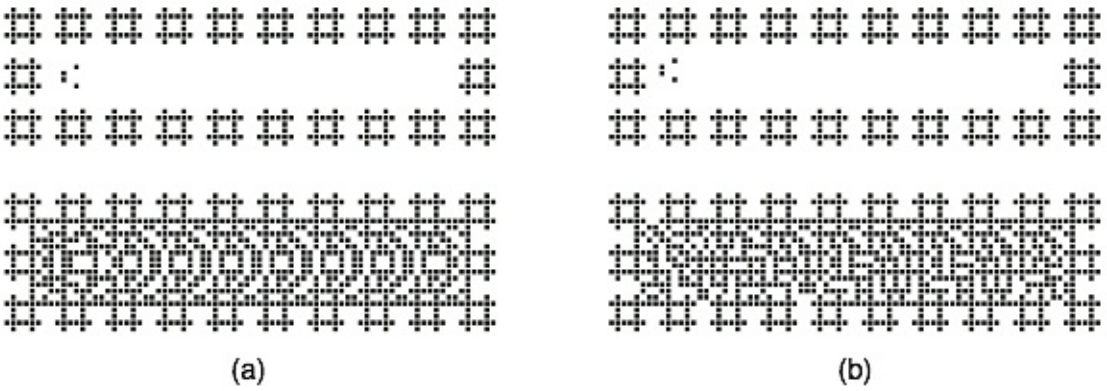

Figura 6: Juárez Martínez Genaro, *Tipos de patrones de propagación en los canales de información (abajo) y sus configuraciones iniciales simulados por móvil auto-localizado (arriba)*. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/Diffusion_Rule/Life_B2-S23456.html.

formación (down/abajo) y sus configuraciones iniciales estimulado por un móvil auto-localizado (top/arriba).

Las señales son representadas por dos tipos de patrones de propagación a lo largo de los canales: a) El valor lógico 0 es representado por frentes de onda regular y b) el valor 1 lógico por patrones menos regulares. La computación ocurre cuando los patrones de propagación en los canales interactúan con las uniones de los canales.

Implementación de la puerta mayoritaria (majority) se muestra en la siguiente figura. La puerta tiene tres entradas: los canales *Norte*, *Oeste* y *Sur* y una *salida* que es el canal *Este*. Esta compuerta fue construida en sus tres patrones de propagación, golpeando en el centro de sus canales, por lo tanto, el resultado final (out) es el resultado de las colisiones de las olas.

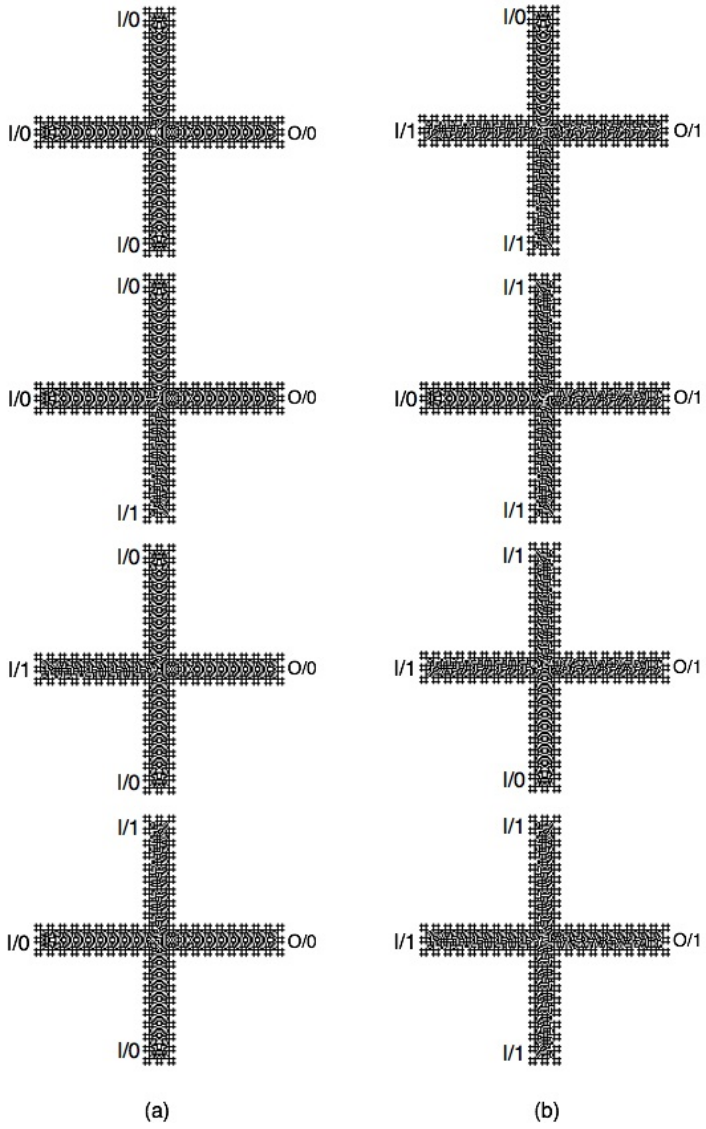

Figura 7: Juárez Martínez Genaro, *Autómata celular implementación de la compuerta mayoritaria: (a) todos los casos corresponden a la entrada mayoritaria con valor de entrada* 0*, (b) todos los casos corresponden a la entrada mayoritaria con valor de entrada* 1. Imagen tomada del sitio: http://uncomp.uwe.ac.uk/genaro/Diffusion_Rule/Life_B2-S23456.html.

La computación en el medio 2c22 está basado en la competencia entre dos tipos de patrones de propagación. Se cree que los resultados teóricos discutidos van a demostrar su valor práctico en la implementación experimental de restricciones geométricamente no lineales de medios procesadores.

9. Conclusiones

Las expresiones y las gramáticas, estas dos formas para representar o describir a los lenguajes son equivalentes, y una nos puede llevar a la otra, basta decidir cual representación se desea utilizar para el manejo de los lenguajes.

Las limitaciones o alcance de cada lenguaje, hacen que algunos de estos contengan a otros, ya forma muy obvia observamos que los *lenguajes regulares* están contenidos en los *lenguajes libres de contexto*, y estos a su vez están contenidos en los *lenguajes sensibles del contexto* y finalmente estos están dentro del conjunto mayor que son los *lenguajes recursivos*, por lo tanto, podemos afirmar que si un lenguaje es recursivamente enumerable, también será sensible al contexto, además de ser libre de contexto y por tanto será también regular.

Por otra parte, a pesar de que los lenguajes más complejos contiene a los más simples, y por ende la relación descrita arriba, no siempre se cumple la relación de manera inversa, es decir que los lenguajes más simples contengan a los más complejos, por ejemplo no todos los lenguajes regulares son lenguajes libres de contexto, ni tampoco todos los libres de contexto son sensibles del contexto, y así sucesivamente.

De forma paralela se esta realizado un artículo con la otra temática, que se mencionó al inicio de este artículo, el cual se refiere a los autómatas, así que se puede complementar ese información con este artículo y con ello poder tener una mejor comprensión de los procesos computables y además sirve como un material de apoyo a los jóvenes universitarios que cursan materias relacionadas con la *teoría de la computación*.

Como trabajo futuro será realizar la equivalencia entre los lenguajes formales y sus respectivas máquinas abstractas ya que ambos se corresponden y de esta manera se estaría enriqueciendo dichos temas y por lo tanto se tendrá un mejor material de apoyo dentro de la teoría de la computación.

Agradecimientos

El autor agradece al COCYTIEG y a la AMC el apoyo económico brindado, sin el cual no hubiera sido posible asistir al XXI verano de la investigación científica para desarrollar este artículo. Agradece nuevamente a la AMC por aceptar al autor y de esta manera poder participar en dicho verano, finalmente al Dr. Genaro Juárez Martínez quien guío y revisó el manuscrito.

Referencias

[1] J. Hopcroft, J. Ullman.- Introducción a la Teoría de Autómatas, Lenguajes y Computación, Addison Wesley, 1979.

[2] D. Kelley.- Teoría de Autómatas y Lenguajes Formales, Prentice Hall Hispanoamericana, 1995.

[3] R. Brena.- Autómatas y lenguajes, Instituto Tecnológico y de Estudios Superiores de Monterrey, 2003.

[4] G. Brookshear.- Teoría de la Computación, Addison Wesley Iberoamericana, 1993.

[5] I. Navarrete, A. Cárdenas.- Teoría de Autómatas y Lenguajes Formales, Universidad de Murcia, 2000.

[6] D. Castro Esteban.- Teoría de Autómatas, Lenguajes Formales y Gramáticas, Universidad de Alcalá, 2004.

[7] T. Toffoli Non-Conventional Computers, Encyclopedia of Electrical and Electronics Engineering (J. Webster ed.), Wiley & Sons (1998).

[8] G. Juárez Martínez.- Introducción a computación no-convencional y autómata celular, University of the West of England Bristol, United Kingdom, 2008.