35

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ACATLÁN

MATEMÁTICAS APLICADAS Y COMPUTACIÓN

SISTEMAS DE LINDENMAYER

Tesis que presenta:

Libertad Rivera Larqué

Para obtener el grado de

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

2001

Sistemas de *Lindenmayer*

Alumna:
Libertad Rivera Larqué
Universidad Nacional Autónoma de México
Escuela Nacional de Estudios Profesionales, Acatlán

Asesor:
Dr. Sergio V. Chapa Vergara
Universidad Nacional Autónoma de México
Escuela Nacional de Estudios Profesionales, Acatlán

19 de Agosto. 2001

Introducción

La teoría de lenguajes formales y la vida artificial son sin duda áreas muy importantes dentro del estudio de la computación. Las matemáticas se presentan también como una poderosa herramienta en este contexto. La aplicación de estas áreas a otros campos como es el de la biología parece tomar más fuerza cada día, sin embargo la dificultad de plantear un modelo matemático que refleje con claridad el proceso de evolución de un sistema no es tarea fácil.

Muchos procesos de crecimiento de vida vegetal pueden ser modelados con sistemas discretos los cuales tienen un manejo mucho más fácil en la generación de vida artificial. Los sistemas de Lindenmayer presentan diversas ventajas sobre otros modelos, ya que éstos son generalmente modelos muy simples que toman el poder de los morfismos iterados, la graficación con autosimilitud y las propiedades de las gramáticas lineales, independientes y sensibles al contexto. Los avances en la geometría fractal y en la graficación permiten presentar a los sistemas de Lindenmayer como poderosos generadores de vida artificial con fundamentos matemáticos sencillos para el estudio de la belleza de las plantas, además de promover el estudio del lenguaje, pues conceptos como el de lingüística, gramática, alfabeto,etc: toman una real importancia.

Los sistemas de Lindenmayer son modelos matemáticos basados en la teoría de lenguajes formales Al variar la forma de reescritura secuencial en las gramáticas chomskianas por una reescritura en paralelo, permiten simular elementos de la naturaleza como nubes, montañas y galáxias además de procesos de crecimiento, florecimiento, metamorfosis, mutaciones, etc; de plantas reales. Estos sistemas presentan además interesantes patrones de autosimilitud estrechamente relacionados con la geometría fractal de la teoría del Caos.

El objetivo principal de esta tesis es establecer la caracterización de los diferentes tipos de sistemas de Lindenmayer dentro del contexto de la teoría de lenguajes formales y exponer de forma precisa el funcionamiento de estos para generar vida artificial.

El presente trabajo titulado *Sistemas de Lindenmayer* se encuentra dividido en cuatro partes:

- El capítulo 1 proporciona un contexto histórico y desglosa una breve introducción a los conceptos básicos relacionados con los sistemas de Lindenmayer y la vida artificial.

- El capítulo 2 introduce a los conceptos teóricos de la teoría de lenguajes formales, enfocándose en describir el funcionamiento y principales propiedades de los dos dispositivos más importantes de esta teoría, los de reconocimiento (*autómatas finitos*) y los generativos (*gramáticas*).

- El capítulo 3 muestra las características y funcionamiento de los principales modelos para generación de vida artificial, basados en dispositivos generativos, estos son los *sistemas de reescritura*.

- El capítulo 4 establece una caracterización de los diferentes tipos de sistemas de Lindenmayer sin ambigüedades.

Finalmente se dan las conclusiones derivadas de este trabajo.

En la medida de lo posible, a través de todo el trabajo, se usan imágenes para clarificar los conceptos y ejemplos.

El escrito fue realizado en LaTeX un derivado del formato TeX sobre el sistema operativo NeXTSTEP v3.3 y compatible con la versión de OPENSTEP, en las instalaciones del Centro de Investigación y Estudios Avanzados del IPN. Departamento de Ingeniería Eléctrica, Sección de Computación.

Contenido

Capítulo 1

Marco Teórico

En 1968 el biólogo Aristid Lindenmayer introdujo un formalismo para la modelación y simulación de organismos multicelulares, que posteriormente fueron llamados *Sistemas de Lindenmayer* o *Sistemas L*. La figura 1.1 muestra de forma simple está relación.

Desarrollo de organismos multicelulares	$\Rightarrow$	Simulación y modelado con Sistemas de Lindenmayer

Figura 1.1: Relación entre el desarrollo y la simulación de organismos vivos.

Este formalismo está relacionado por las gramáticas, de la teoría de lenguajes formales. Ésta, es un área interdisciplinaria que permite definir gramáticas formales o lenguajes específicos en muchas disciplinas científicas, además ha sido enriquecida con importantes aportaciones de la teoría matemática y ha tenido un gran interés por parte de muchos científicos en todo el mundo desde su surgimiento.

A partir de un desarrollo riguroso de la teoría de los sistemas L, siguió la aplicación de esta teoría a la modelación de plantas, una de las más importantes fue a las gráficas por computadora que permitieron visualizar estructuras en formas ordenadas de diagramas esquemáticos dándoles interpretaciones tridimensionales reales de plantas. Estas visualizaciones marcaron una relación importante, y a la vez sorprendente, con los fractales.

La evolución de los organismos puede ser representada por medio de las gramáticas conocidas en la teoría de lenguajes formales éstas últimas son llamadas gramáticas Chomskianas o de Chomsky. La diferencia principal entre las gramáticas de Chomsky y las de Lindenmayer radica en el método de aplicar las reglas de producción. En las gramáticas de Chomsky las producciones son aplicadas en forma secuencial en cada paso mientras que en los sistemas de Lindenmayer se aplican en *paralelo* y *simultáneamente* se reemplazan todas las letras de una palabra.

Esta diferencia tiene gran importancia en las propiedades de los sistemas de Lindenmayer, porque hay lenguajes que pueden ser generados por sistemas de Lindenmayer independientes del contexto y no por las gramáticas independientes del contexto de Chomsky, de esta forma los sistemas de Lindenmayer han sido considerados como un modelo formal para el desarrollo de plantas además de tener propiedades poderosas para la generación de fractales.

La importancia del poder de los sistemas de Lindenmayer para generar imágenes, radica en la operación en paralelo, como forma de reescritura.

La figura 1.2 muestra la gráfica de un sistema de Lindenmayer y algunos ejemplos de gráficas de fractales.

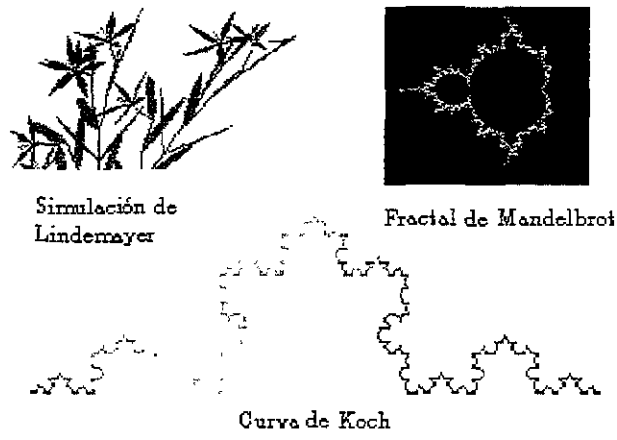


Figura 1.2: Relación de los sistemas L con los fractales.

En general pueden mencionarse dos características principales de los sistemas de Lindenmayer:

- *Paralelismo en los Procesos de Reescritura.*
  Debido a que en los lenguajes que se utilizan para modelar desarrollos biológicos, los organismos cambian simultáneamente.

- *Noción de Gramática.*
  Las gramáticas se utilizan como una descripción de un proceso dinámico más que estático.

Esta última característica dio inicio a un extenso estudio de las secuencias generadas vistas como *secuencias* en lugar de ser simples conjuntos.

El desarrollo de una planta puede ser descrito, como lo muestra la sencilla figura 1.3, por un sistema de reescritura que reemplaza repetidamente *ancestros* por configuraciones *descendientes*, de forma que las reglas de reescritura especifican *como* reemplazar un predecesor por una configuración de cero, uno o más sucesores.

Ancestros ⇒ Reglas de reescritura ⇒ descendientes

Figura 1.3: Relación ancestros-descendientes.

1.1 Alfabetos, cadenas y lenguajes

Un *alfabeto* es un conjunto Σ finito no vacío de símbolos. a partir de estos símbolos individuales se construyen cadenas. las cuales son *secuencias finitas* de símbolos del alfabeto. En general las letras de un alfabeto se denotan con:

$$a, b, c \ldots$$

y los nombres de cadenas como.

$$u, v, w \ldots .$$

Por ejemplo. sea el alfabeto·

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

con el que pueden construirse las cadenas:

$$aaaa, ababaabb, bbbbb, a, b$$

La *cadena vacía* se define como una cadena que no tiene símbolos y se denota por λ. las siguientes relaciones definen mejor a esta cadena.

$$\begin{array}{lll} |\lambda| = & 0 & \\ \lambda w = & w\lambda = & w \\ \lambda\lambda = & \lambda & \end{array}$$

Aunque estas relaciones aparentemente son muy fáciles, pueden llegar a comprenderse mal si se piensa que λ es un símbolo que pertenece al alfabeto. cuando es en realidad una cadena. que representa a la identidad en la concatenación.

La *concatenación* de dos cadenas w y v es la cadena que se obtiene al juntar los símbolos de v a la derecha de w.

$$\begin{array}{l} w = a_1 a_2 a_3 \cdots a_n \\ v = b_1 b_2 b_3 \cdots b_m \end{array}$$

la concatenación de w y v se denota como wv.

$$wv = a_1 a_2 a_3 \cdots a_n b_1 b_2 b_3 \cdots b_m$$

La concatenación de palabras es análoga a la multiplicación de los números reales. en su relación de longitud y logaritmo (no en conmutatividad) Dados dos reales positivos x y y se tiene.

$$\begin{array}{lll} log(x * y) = & logx + logy & \\ log(x * 1) = & logx + log1 = & logx \end{array}$$

y con dos palabras x y y se obtiene:

$$\begin{array}{lll} |xy| = & |x| + |y| & \\ |x\lambda| = & |x| + |\lambda| = & |x| \end{array}$$

La *reversa* de una cadena se obtiene cuando se escriben los símbolos en *orden inverso*. si w es la cadena anterior, entonces la reversa de w es:

$$w^R = a_n \cdots a_3 a_2 a_1$$

Una definición recursiva de w^R se da de la siguiente forma. Para todas las palabras w sobre un alfabeto Σ. w^R es definida recursivamente por:

1. λ si $x = \lambda$
2. $x^R = a$, si $w = ax$ para alguna $a \in \Sigma$ y $x \in \Sigma^*$

Por ejemplo, si $x = amor$ sobre el alfabeto Σ. Entonces x^R

$$\begin{aligned} (amor)^R &= (mor)^R a \\ &\quad (or)^R ma \\ &\quad (r)^R oma \\ &\quad \lambda^R roma \\ &\quad roma \end{aligned}$$

Cualquier cadena consecutiva de caracteres en una palabra w, es una *subcadena* de w. Si

$$w = vu$$

las subcadenas v y u son el *prefijo* y el *sufijo* de w. Una propiedad simple de las cadenas es:

$$\mid uv \mid = \mid u \mid + \mid v \mid$$

Si w es una cadena entonces w^n implica repetir w, n veces. Un caso especial es:

$$w^0 = \lambda$$

Si Σ es un alfabeto. entonces Σ^* denota el conjunto de cadenas obtenidas por la concatenación de cero o más símbolos de Σ, es el *lenguaje universal*.

Por ejemplo, sea el alfabeto definido por:

$$\Sigma = \{a\}$$

entonces el lenguaje universal está compuesto por las palabras:

$$\begin{array}{l} \lambda \\ a \\ aa \\ aaa \\ aaaa \\ aaaaa \\ \vdots \end{array}$$

El conjunto Σ^* siempre contiene a λ. Para excluir a la cadena vacía se define:

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

El conjunto Σ es finito y los conjuntos Σ^* y Σ^+ son siempre infinitos ya que la longitud de las cadenas no está limitada en estos conjuntos.

En la figura 1.4 se representan los conjuntos Σ, Σ^* y 2^{Σ^*}. A partir del conjunto Σ (que es un alfabeto finito), se pueden empezar a formar cadenas aplicando la operación de concatenación a los símbolos del alfabeto y de esta forma se empieza generar el conjunto Σ^* (que es un conjunto de cadenas). En este nuevo conjunto, se puede seguir aplicando la operación de concatenación, concatenando unas cadenas con otras y se generarán nuevas cadenas que pertenecen al mismo conjunto Σ^*. Cuando se toman subconjuntos de estas cadenas se genera el conjunto 2^{Σ^*}, que es el conjunto de lenguajes (o conjunto potencia).

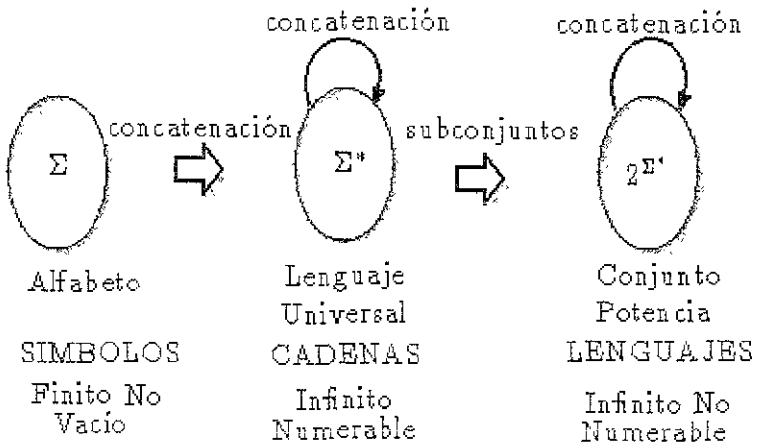


Figura 1.4: Relación entre alfabetos, cadenas y lenguajes.

1.2 Autómatas

Un autómata es un modelo abstracto de una computadora digital. Cada autómata tiene sus propias características.

En general, tienen un *mecanismo de lectura* para la entrada, tal que la entrada es una cadena sobre algún *alfabeto* escrito en un archivo de entrada que el autómata puede leer pero no modificar. El archivo de entrada a su vez está dividido en celdas, en cada una de las cuales se pone un *símbolo*. El mecanismo de entrada también puede dectectar el fin de la cadena de entrada. El autómata puede tener un *dispositivo de almacenamiento* temporal, que consiste en un número ilimitado de celdas. El autómata tiene una *unidad de control*, la cual puede tener uno de cualquier número finito de estados internos y se puede pasar de un estado a otro de forma definida.

La figura 1.5 ilustra las partes de un autómata y la forma en que éstas están relacionadas.

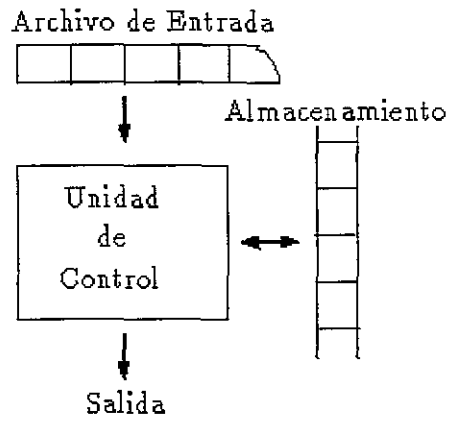


Figura 1.5: Partes de un autómata.

Un autómata opera en *tiempos discretos*. En cualquier tiempo, la unidad de control está en un estado interno y el mecanismo de entrada está leyendo un símbolo particular del *archivo de entrada*, el estado interno de la unidad de control en el siguiente paso estará determinado por la *función de transición*. Esta función de transición proporciona el siguiente estado en términos del estado actual y el símbolo de entrada actual.

Este modelo general, cubre todos los modelos de autómatas, aunque se hace una distinción importante entre los autómatas finitos determinísticos y los no determinísticos. donde los primeros se caracterizan por tener bien definida la conducta de sus transiciones. mientras que los segundos, tienen un conjunto de posibles opciones de conducta.

Un autómata que su salida está restringida a responder únicamente si o no, es llamado *aceptador* . esto es, dada una cadena de entrada. el aceptador decide si se acepta o se rechaza. Como se muestra en la figura 1.6.

Figura 1.6: Aceptador.

1.3 Gramáticas

Para poder estudiar los lenguajes matemáticamente, se pueden utilizar las gramáticas para describirlos de manera formal.

Una *gramática* es una colección de símbolos no terminales y terminales, junto con un símbolo de inicio y un conjunto finito de reglas de reescritura. Para diferenciar la colección de símbolos, los no terminales se encierran entre corchetes de la siguiente forma:

<no terminal>.

uno de estos no terminales se considera como *símbolo de inicio*. Los términos que no aparecen entre corchetes son los *terminales*. Cada una de las producciones de la gramática se denominan *regla de reescritura* y consisten de una parte izquierda y una derecha conectadas por una flecha. En general, los lados derecho e izquierdo de las reglas de reescritura de una gramática pueden ser cualquier combinación de terminales con no terminales.

Una *gramática* es una cuádrupla

$$G = (V, T\ S, P)$$

donde

V	es un conjunto finito de objetos llamados *variables*.
T	un conjunto finito de objetos llamados símbolos *terminales*.
$S \in V$	es un símbolo inicial llamado variable de *inicio*.
P	es un conjunto finito de *producciones*.

De tal forma que $V \neq \emptyset$, $T \neq \emptyset$ $V \cap T = \emptyset$.

Por ejemplo, una gramática para nuestro lenguaje que indique si un enunciado está *bien formado* o no, puede ser formada con las siguientes reglas de reescritura.

<enunciado>→<sujeto><predicado><punto>
<sujeto>→<sustantivo>
<sustantivo>→Marco
<sustantivo>→Patricia
<predicado>→<verbo><objeto>
<verbo>→quiere
<objeto>→a<sustantivo>
<punto>→.

El proceso de derivación de una cadena, (o cuando la gramática puede generar a la cadena) empieza con el símbolo de inicio y produce la cadena sustituyendo sucesivamente los patrones que se encuentran en el lado izquierdo de las reglas de reescritura de la gramática con las expresiones de la derecha hasta que sólo queden terminales.

Sea el siguiente ejemplo:

<enunciado>
⇓
<sujeto><predicado><punto>
⇓
Patricia<predicado><punto>
⇓
Patricia<verbo><objeto><punto>
⇓
Patricia quiere a <sustantivo><punto>
⇓
Patricia quiere a Marco<punto>
⇓
Patricia quiere a Marco.

Si los símbolos terminales de una gramática G, son símbolos de un alfabeto Σ, entonces se dice que:

G es una gramática del alfabeto Σ, que especifica a un lenguaje de Σ que consiste en las cadenas generadas por G.

A este lenguaje se le representa con:

$$L(G)$$

Esta es una idea general del concepto, comenzando desde el nivel más alto <enunciado> (símbolo de inicio) y reduciendo sucesivamente, se van construyendo los bloques del lenguaje.

Las *reglas de producción* son la parte más importante de las gramáticas porque describen la forma en que ésta transforma una cadena en otra y definen a su lenguaje asociado. Si se escriben las reglas de producción de la forma:

$$x \rightarrow y$$

las producciones se pueden aplicar de la siguiente forma:

$$w = u\mathbf{x}v$$

aplicando la producción $x \rightarrow y$ a la cadena. se obtiene la cadena:

$$z = u\mathbf{y}v$$

y esto se denota como:

$$w \Rightarrow z$$

esto es, w *deriva a* z o z es *derivada de* w. Si

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

denota que w_1 deriva a w_n y puede escribirse como:

$$w_1 \Rightarrow^* w_n$$

donde * indica un número no especificado de pasos (incluyendo cero), es decir en cero o más pasos.

Si se tiene la cadena wA y existen varias producciones de A en G, entonces la cadena puede ser reescrita de varias formas y el proceso de reescritura es *no determinístico*. Se puede usar una producción si ésta es aplicable y puede usarse todas las veces que sea necesaria. Al aplicar las reglas de producción en un orden distinto se derivan muchas cadenas y el conjunto de todas estas cadenas posibles, es el *lenguaje generado* por la gramática.

1.4 Sistemas de Lindenmayer

Los sistemas de Lindenmayer han sido considerados como una teoría matemática de las plantas, que da un énfasis a las relaciones que hay entre las células (o módulos) de los organismos. Además, promueve el estudio de la belleza de las plantas que ha atraído la atención de los matemáticos durante siglos. La figura 1.7 muestra ejemplos de esta belleza.

Figura 1.7: Sistemas de Lindenmayer.

El propósito original de los sistemas de Lindenmayer fue la modelación del desarrollo de organismos simples. Las construcciones matemáticas aplicadas a estos organismos fueron formuladas originalmente como arreglos celulares de autómatas finitos, siendo motivados por los modelos de Redes Nerviosas de McCulloc y Pitts y los autómatas de autorreproducción de von Neumann.

Si se asume un organismo vivo como un autómata finito este puede estar en uno de un número finito de estados, el tiempo en cada uno de los cambios de estado avanza en pasos discretos. En cada paso de tiempo el siguiente estado de cada célula corresponde al estado, determinado por la función de transición.

Una célula en cualquier estado puede cambiar a:

- una sola nueva célula.
- ser eliminada (o morir).

- dividirse en una cadena de células,
- o puede permanecer sin cambio.

Además este nuevo estado puede depender ya sea del estado actual o pueden influir los estados de las células vecinas.

Por otro lado, el nuevo estado puede ser un sólo nuevo estado (*determinismo*) o más de un nuevo estado. (*indeterminismo*). Los sistemas puenden tener como característica en la función de transición que ninguna célula muera y esto hará que el sistema sea *propagado.*

Un *Sistema de Lindenmayer* puede ser visto como una construcción matemática que consiste de un conjunto de reglas de producción con o sin interacción entre sus células que se aplican en paralelo, un alfabeto de símbolos y un axioma o estado inicial.

Por ejemplo, sea el sistema de Lindenmayer definido por .

1. *Conjunto de Reglas de Producción:*
   - $a \rightarrow a$
   - $b \rightarrow ab$
2. *Alfabeto:*
   - $\{a, b\}$
3. *Axioma:*
   - ab

Entonces aplicando las reglas de producción en paralelo, a partir del axioma ab, se obtienen la secuencia de palabras:

$$\begin{array}{c} < ab > \\ \Downarrow \\ < a^2b > \\ \Downarrow \\ < a^3b > \\ \Downarrow \\ \vdots \\ \Downarrow \\ < a^nb > \end{array}$$

y puede decirse que este sistema de Lindenmayer produce las palabras:

$$SL = \{a^n b \mid n \geq 1\}$$

La graficación de estos modelos matemáticos da origen a las *plantas virtuales*, éstas son simulaciones del crecimiento y desarrollo estructural de las plantas reales. Las simulaciones utilizan reglas de desarrollo expresadas en sistemas de Lindenmayer. Cuando las reglas son aplicadas recursivamente o representan sistemas en tres dimensiones, la simulación aumenta notablemente su complejidad, como lo muestra la figura 1.8.

Figura 1.8. Reescritura de sistemas de Lindenmayer en 3D.

La *vida artificial* es usualmente comprendida como el estudio de construcciones hechas por el hombre que exhiben en algún sentido o en algún aspecto la vida de organismos vivos existentes. Extiende la biología tradicional que se refiere a las cadenas de carbón de vida evolucionadas en la Tierra. Trata de sintetizar la vida como conducta dentro de las computadoras y otros componentes.

Una *planta virtual* se genera de un modelo que contiene reglas para la formación de nuevas partes de la planta, cambios en el tamaño y en el contorno. las reglas de reescritura pueden expresar más allá que el desarrollo estructural o la forma de crecimiento como son, procesos de marchitación, tropismos, degeneración física, etc. Pueden ser restringidas a los cambios observados en un medio ambiente constante u otras condiciones. La figura 1.9 muestra algunos ejemplos de estas simulaciones.

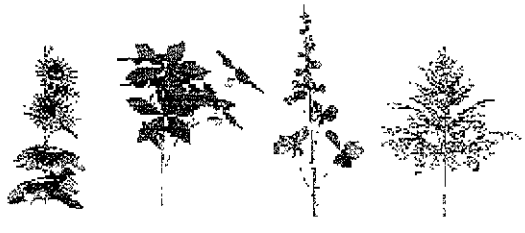


Figura 1.9: Plantas virtuales con sistemas de Lindenmayer

El uso de simulación para plantas virtuales, puede hacerse para simular plantas completas o partes de un sistema. Pueden ser representaciones de plantas reales, cuyas reglas de producción se obtienen mediante la observación y la medición o bien pueden ser hipotéticas.

Las plantas virtuales no son simples imágenes generadas por computadora, sino que cada imagen representa un modelo de un planta en un instante particular.

Este desarrollo ocurre en paralelo en cualquiera de estos organismos. Por lo tanto el *paralelismo* fue una característica primordial de los sistemas de Lindenmayer.

Esto significa desde el punto de vista de reescritura, que todo debe ser reescrito en un sólo paso del proceso de reescritura, en contraste con la reescritura secuencial de las gramáticas estructuradas: sólo una parte específica de la palabra es reescrita en cada paso.

El paralelismo no puede ser siempre simulado por la reescritura secuencial.

Como ejemplo, supóngase un sistema cuya única regla es:

$$a \to a^2$$

y tiene el axioma:

$$a^3$$

Lo que se obtiene si la reescritura es *secuencial*, es que se puede reemplazar a en un paso por a^2, obteniendo eventualmete todas las palabras

$$a^i, \text{ tal que } i \geq 3.$$

Si la reescritura es en *paralelo*, la palabra a^6 resulta en un solo paso. No es posible obtener a^4 o a^5 de a^3. Entonces sólo se tienen las palabras:

$$a^{3 \cdot 2^i}.$$

De tal forma que se obtiene el lenguaje:

$$\{a^{3 \cdot 2^i} \mid i \geq 0\}$$

La *regla falsa* $a \to a$ se comporta muy diferente entre reescritura secuencial y paralela.

En la reescritura secuencial, ésta no tenía influencia y podía ser omitida. Sin embargo en la reescritura en paralelo la regla $a \to a$ hace posible la simulación de reescritura secuencial: la regla real $a \to a^2$ es aplicada a una ocurrencia de a y la regla falsa $a \to a$ se aplica para que permanezcan las ocurrencias.

Consecuentemente todas las palabras a^i, $i \geq 3$, son obtenidas por la reescritura en paralelo a partir de a^3, si ambas reglas $a \to a^2$ y $a \to a$ están disponibles.

Un punto importante es la predicción del futuro de los sistemas de Lindenmayer. Los sistemas de Lindenmayer fueron introducidos principalmente para modelar el desarrollo de organismos multicelulares, esto es de alguna forma el desarrollo de vida real. Sin embargo hay numerosas aplicaciones en gráficas de computadoras, donde estos sistemas han sido utilizados para describir formas de vida imaginaria, como jardines de vida.

Capítulo 2

Teoría de Lenguajes Formales

La Teoría de Lenguajes Formales es parte, entre otras, de las Ciencias de Computación Teórica y Lingüísticas Computacionales.

La primera, estudia los conceptos fundamentales y métodos de las ciencias de la computación con herramientas teóricas, así los modelos teóricos estudian y clarifican conceptos de las ciencias de la computación. El estudio de estos modelos y el desarrollo de herramientas teóricas, tomadas de las matemáticas y la lógica, se usan fundamentalmente para dar una estructura de unificación a la práctica. La figura 2.1 muestra los campos de interés de la Computación Teórica, estos campos están estrechamente relacionados.

Teoría de Lenguajes Formales	Teoría de Complejidad
Teoría de Autómatas	Teoría de Semánticas
Teoría de Computabilidad	Aproximaciones Matemáticas

Figura 2.1: Campos de la computación teórica.

- **Teoría de Lenguajes Formales.**
  Surgieron después de la introducción del concepto de gramáticas en la ciencia de la Computación. Ha sido exitosa en la clasificación de clases de lenguajes y gramáticas, ya sea por propiedades de las reglas gramaticales, por propiedades de parseo o por propiedades de complejidad.

- **Teoría de Autómatas.**
  Comenzó antes que la teoría de lenguajes formales. Realiza diferentes versiones y modelos de máquinas, máquinas de Turing y autómatas para simular la actividad del cerebro. La entrada y salida del autómata pueden considerarse como cadenas de símbolos (o enunciados) de un lenguaje.

- **Teoría de Computabilidad.**
  Sugió como un campo de la lógica. Sus primeros componentes fueron la teoría de funciones recursivas y la máquina de Turing como modelo de un algoritmo. Estudia las limitaciones (teóricas) de la ciencia

de la computación. Muestra lo que se puede realizar y no, estableciendo propiedades fundamentales de recursividad y de conjuntos recursivamente enumerables.

- **Teoría de Complejidad.**
  Es el estudio teórico de los conceptos que pueden ser usados para medir qué tan efectivo es un algoritmo y trata de encontrar técnicas eficientes para resolver problemas. Estas medidas se dan en términos del gasto de los recursos de la computadora (tiempo y memoria) en modelos de máquinas específicas (máquinas de Turing o máquinas de acceso aleatorio). Así como la teoría de computabilidad trata de decidir si un problema es soluble o no, la teoría de complejidad, responde a la pregunta si hay una solución que sea realizable prácticamente.

- **Teoría de Semánticas.**
  Se refiere al desarrollo de sistemas formales para describir el significado de los constructores de lenguajes de programación. Los métodos principales de descripción de semánticas son la aproximación operacional y la aproximación matemática. Esta teoría está basada en el cálculo lambda de A. Church y en los modelos de su cálculo proporcionado por Scott.

- **Aproximaciones Matemáticas.**
  Estas se han dado principalmente a la construcción de Compiladores, de Base de Datos, Procesos Paralelos, Inteligencia Artificial, etc.

La tabla 2.1 muestra las áreas de aplicación y cada uno de sus precursores, en el campo de la teoría de lenguajes formales.

Área	Precursores
Lógica, Teoría de Funciones Recursivas	Post Church.Turing
Neurofisiología	McCulloch, Pitts
Lingüística	Chomsky
Construcción de Compiladores	Backus, Naur, Floyd
Desarrollo Biológico	Lindenmayer

Tabla 2.1: Áreas de aplicación de la teoría de lenguajes formales.

Algunos de estos orígenes se caracterizan como la aplicación de lógica en intentos para formalizar la manipulación de símbolos en ciertas áreas.

Los conceptos de la teoría de lenguajes formales se aplicaron a la biología, cuando Aristid Lindenmayer estudió modelos matemáticos para el desarrollo de plantas. Basado en la teoría de autómatas describió el desarrollo de organismos multicelulares con una colección de autómatas finitos interactuando, lo que posteriormente se dio como una estructura gramatical para representar los sistemas de Lindenmayer. Los símbolos de una cadena son reescritos simultáneamente y existen reestricciones en la aplicación de las reglas.

2.1 Lenguajes Regulares

Las máquinas teóricas conocidas como autómatas finitos aunque tienen un poder limitado, son capaces de reconocer numerosos patrones de símbolos que son los que pertenecen a la clase de *lenguajes regulares*.

Como lo muestra la siguiente tabla 2.2 los autómatas finitos se encuentran en el nivel más bajo de la jerarquía de las principales máquinas y lenguajes que estudia la teoría de lenguajes formales.

Maquinas	**Lenguajes**
Máquina de Turing	Lenguajes Estructurados por Frases
Autómatas de Pila	Lenguajes Independientes del Contexto
Autómatas Finitos	Lenguajes Regulares

Tabla 2.2: Jerarquía de máquinas y lenguajes.

Esta base establece la importancia de tales máquinas y lenguajes desde una perspectiva teórica, pero su importancia no se limita a la teoría. Los conceptos que presentan los autómatas finitos y los lenguajes regulares son de interés fundamental para la mayoría de las aplicaciones que requieren técnicas de reconocimiento de patrones.

Una de estas aplicaciones es la construcción de compiladores. Por ejemplo un compilador debe de ser capaz de reconocer cuáles son las cadenas del programa fuente que deben considerarse como representaciones de objetos individuales, es decir, nombres de variables, constantes numéricas y palabras reservadas. Esta tarea de reconocimiento de patrones es manejada por el analizador léxico del compilador.

2.1.1 Máquina Abstracta

Un concepto básico para dar inicio al estudio de los autómatas finitos es el de *máquina abstracta* como lo muestra la figura 2.2.

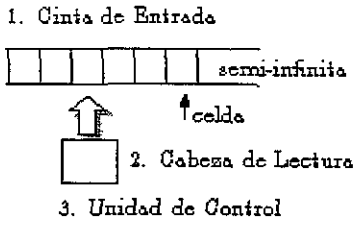


Figura 2.2: Representación de una máquina abstracta.

Dado un lenguaje se construye una *máquina abstracta* particular que verifica si una cadena pertenece a ese lenguaje o no.

La figura 2.3 muestra como al tomar cualquier cadena del lenguaje universal la máquina abstracta verifica si ésta pertenece al lenguaje.

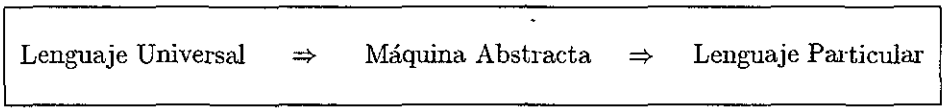


Figura 2.3: Verificación de cadenas en una máquina abstracta.

Para construir esta máquina abstracta, se necesitan tres elementos fundamentales:

1. *Cinta de entrada.*
2. *Cabeza de lectura:*
   - con las operaciones:
     - Lectura.
     - Escritura.
3. *Unidad de Control:*
   - con las operaciones:
     - Controlar la lectura.
     - Controlar el movimiento de la cabeza de lectura.

Entonces la máquina abstracta puede ser vista como la figura 2.4:

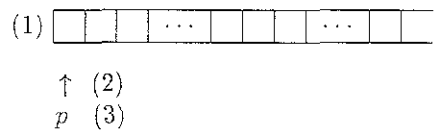


Figura 2.4: Otra representación para una máquina abstracta.

donde (1), (2) y (3) representan la cinta de entrada, la cabeza de lectura y la unidad de control, respectivamente. Además se puede definir la operación de la máquina en las siguientes etapas:

1. *Etapa de Inicio.*

   (a) La cadena a que se va a analizar se coloca símbolo a símbolo en la cinta de entrada como se muestra en la figura 2.5.

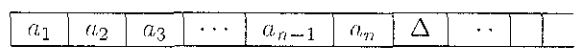


Figura 2.5: Colocación de la cadena en la cinta de entrada.

   (b) La cabeza de lectura se coloca sobre la celda del extremo izquierdo como se muestra en la figura 2.6.

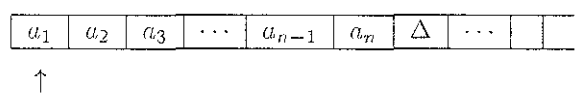
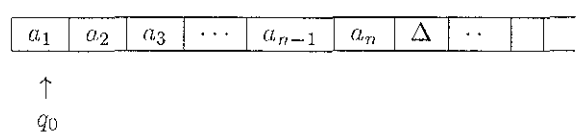


Figura 2.6: Colocación de la cabeza de lectura.

   (c) La unidad de control se ajusta en el estado inicial. Este estado se denomina el estado actual de la máquina.



2. *Etapa de Ejecución.* (Ciclo Básico de Operación)

   (a) Se lee el símbolo que está abajo de la cabeza de lectura, este símbolo se denomina *símbolo actual.* Si no hay símbolo o el símbolo leído corresponde al espacio en blanco, la máquina se detiene.

   (b) Se calcula el nuevo estado de la máquina a partir del símbolo y estados actuales usando la tabla de transiciones. Si el nuevo estado no está definido, la máquina se detiene.

   (c) La cabeza de lectura se mueve una celda a la derecha.

   (d) El nuevo estado se convierte en el estado actual.

3. *Examen de configuración final.*

   (a) Si el estado que queda en la unidad de control es un estado final, la cadena se *acepta* de lo contrario, se *rechaza.*

El ciclo básico de operación se repite. (Se limpia la cinta y se obtiene una nueva cadena.)
La tecnología de las cintas, las cabezas de lectura y las unidades de control con las que se describen los autómatas finitos no es un factor crítico en esta definición, ya que esta implantación es un modelo informal que ayuda a recordar las propiedades de los autómatas finitos. Pueden construirse máquinas con las mismas propiedades de computación utilizando diversas tecnologías y cada una de estas máquinas debe reconocerse como un autómata finito. En otras palabras, la tecnología con la que se implementen estos autómatas finitos no aumentará su poder de reconocimiento.

Para seguir con el análisis de estas máquinas se debe dar una definición formal y precisa de un autómata finito que identifique las características pertinentes de estas máquinas.

2.1.2 Modelo Matemático

Un *Autómata Finito Determnístico* **(AFD)M** se especifica por una quíntupla $M = (Q, \Sigma, \delta, q_0, F)$ donde:

- Q es un alfabeto de *estados.*
- Σ es un *alfabeto de símbolos de entrada.*
- $q_0 \in Q$ es el *estado inicial.*
- $F \subset Q$ es el conjunto de *estados finales.*
- $\delta : Q \times \Sigma \to Q$ es la *función (completa) de transición.*

Se asume además que:

$$Q \cap \Sigma = \emptyset.$$

Esto es debido a que los símbolos que identifican *estados* son diferentes de los símbolos del *alfabeto.*

La notación:

$$\delta : Q \times \Sigma \to Q$$

denota al conjunto de pares ordenados, (donde el primer elemento es un estado y el segundo es un símbolo del alfabeto), que mapean a un estado cualquiera del conjunto de estados Q por ejemplo,

sean a, b símbolos de un alfabeto Σ, esto es $\Sigma = \{a, b\}$
y sea q_0 un estado, de tal forma que $q_0 \in Q$
entonces por la notación anterior, $\delta : Q \times \Sigma \to Q$ puede ejemplificarse con $\delta : (q_0, a) = q_0$.

Esta caracterización matemática implica que en la cinta de entrada sólo se podrán almacenar cadenas con símbolos de Σ.

- El *alfabeto* Σ especifica a los símbolos que se pueden almacenar en la cinta de entrada. Además, Σ es el alfabeto sobre el cual se construye el lenguaje que se quiere reconocer.

- El *conjunto* Q especifica a los estados que caracterizan a la unidad de control.

- El único *estado inicial* es $q_0 \in Q$. un autómata puede tener sólo un estado inicial.

- El subconjunto $F \subset Q$ caracteriza a los *estados finales*.
  Esto significa que el autómata puede tener uno o más estados finales. Un autómata finito que no tiene estados finales no reconoce ningún lenguaje.

- La *función de transición* δ especifica a las instrucciones de operación de la máquina. La función δ representa a la tabla de transición.

Hasta aquí, sólo se han especificado los elementos del autómata (no la operación), porque el modelo matemático es una idealización que permite describir las propiedades del objeto.

Como δ es una función completa, el AFD M se dice que es un *autómata completo.*

A la definición se le hacen variantes y ésto implica tener varias familias de Autómatas Finitos

Sea el AFD $M = (Q, \Sigma, \delta, q_0, F)$. Se dice que una cadena $Q\Sigma^*$ es una *configuración* de M.
Esta cadena representa al estado actual de M y lo que resta de leer de la cadena de entrada, como lo muestra la figura 2.7:

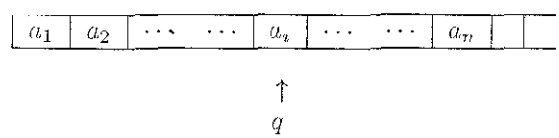


Figura 2 7: Configuración de la máquina M.

donde

$$q \in Q \quad y \quad \Sigma^* = a_i a_{i+1} \ldots a_{n-1} a_n$$

De tal forma que la configuración se lee como

$$q a_i a_{i+1} \ldots a_{n-1} a_n$$

La configuración de la máquina indica que resta por leer como se muestra en la figura 2.8.

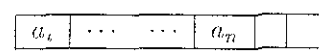


Figura 2.8: Símbolos que restan por leer.

está por iniciar a leer a_i. el estado actual es q. o bien la unidad de control está en el estado q.

De la definición de configuración se desprende que $q_0 x$. donde $x \in \Sigma^*$ es la cadena que se va analizar, corresponde a la *configuración inicial* de M. Si

$$x = a_1 a_2 \ldots a_n.$$

la configuración inicial es:

$$q_0 a_1 a_2 \ldots a_n$$

donde el prefijo $q_0 a_1$ representa al estado y al símbolo actuales de M.

Ahora, $q_0 a_1 a_2 \ldots a_n$ puede representarse en la máquina como en la figura 2.9:

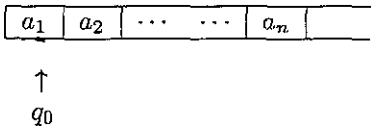


Figura 2.9: Configuración inicial.

De manera general, para una configuración py, donde:

- p es el estado actual,
- $y = b_1 b_2 \ldots b_m$ es la cadena que resta por leer,
- $pb_1 b_2 \ldots b_n$ es la configuración actual.

el prefijo pb_1 representa al estado y símbolos actuales como se muestra en la figura 2.10:

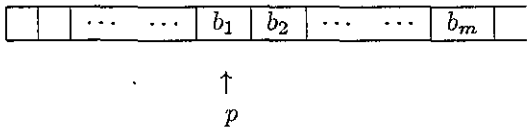


Figura 2.10: Representación del estado y símbolo actual.

No existe memoria hacia atrás, ésta es sólo local, no importa lo que se ha leído sino lo que se va a leer.

Sea el AFD $M = (Q, \Sigma, \delta, q_0, F)$ y sean las configuraciones px y qy. Entonces se puede definir al símbolo $\vdash$ como una relación binaria sobre $Q\Sigma^*$ de la forma:

$$px \vdash qy \text{ si } x = ay, \text{ para algún } a \in \Sigma, \text{ y } \delta(p, a) = q;$$

esto representa un *ciclo básico de operación* de M.

Además puede ser representado gráficamente como se muestra en la figura 2.11. Supóngase la configuración en el estado p, leyendo el símbolo a, tal que $x = ay$.

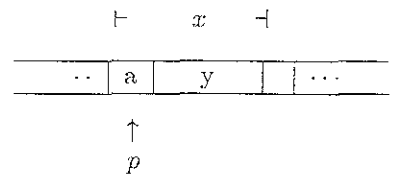


Figura 2.11: Representación de p leyendo a a tal que $x = ay$

o bien como se muestra en la figura 2.12:

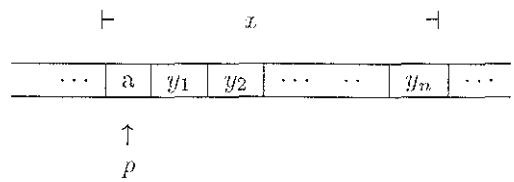


Figura 2.12: Representación de p leyendo a a tal que $x = y_1 \dots y_n$.

Los dos extremos de la cinta están abiertos, porque del lado izquierdo, puede ser una cadena o una subcadena y por el lado derecho se supone una cinta semi-infinita.

Esta representación muestra como al tomar cualquier cadena del lenguaje universal la máquina abstracta verifica si ésta pertenece al lenguaje.

Si $\delta(p, a) = q$ esto es, del estado p con el símbolo a pasa al estado q. Entonces aplicando esta función de transición y recordando que $x = ay$, ocurre un ciclo básico de operación como se muestra en la figura 2.13:

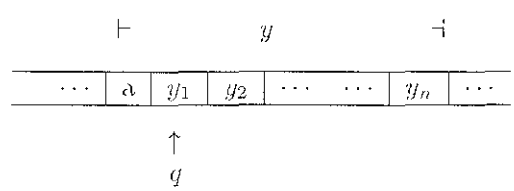


Figura 2.13. Ciclo básico de operación.

Un *ciclo básico de operación* ocurre cuando la cabeza se mueve un lugar hacia adelante y pasa a q pero como $x = ay$, la configuración ya no contiene a a, y por lo tanto ya es y

Si ocurre $px \vdash qy$ en M se dice que la configuración px *produce la configuración* qy.

Para $k \geq 1$ se escribe $px \vdash^k qy$ si $k = 1$ y $px \vdash qy$ o si $k > 1$ y existe una configuración rz tal que $px \vdash rz$ y $rz \vdash^{k-1} qy$: en este caso se dice que ocurrieron k ciclos de operación.

Lo anterior puede ser visto de la siguiente forma:

- $k = 1 \Rightarrow$ un ciclo básico como lo muestra la figura 2.14:

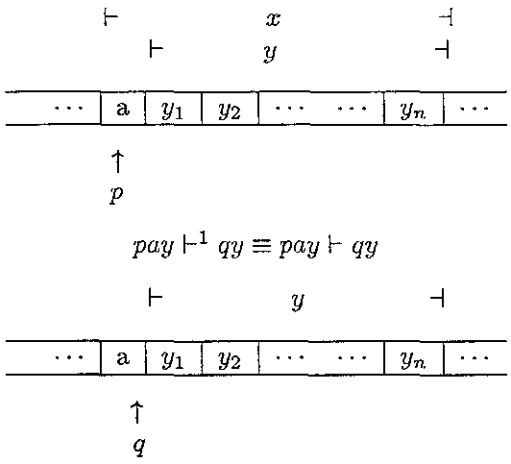


Figura 2.14: Representación de k ciclos básicos cuando $k = 1$.

- $k > 1 \Rightarrow k$ ciclos básicos.

$px \vdash^k qy \equiv px \vdash rz, rz \vdash^{k-1} qy$

Si $x = az$ y la unidad de control se encuentra leyendo el símbolo a en el estado p como lo muestra la figura 2.15:

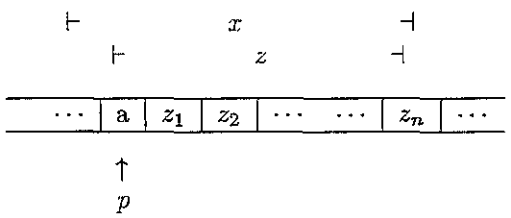


Figura 2.15: Representación de k ciclos básicos cuando $k > 1$.

Aplicando la función de transición $\delta(p, a) = r$ la unidad de control se mueve una celda a la derecha cayendo en el estado r y comenzando a leer la cadena z. en este caso especial al símbolo z_1 como lo muestra la figura 2.16:

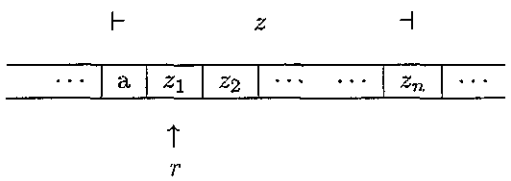


Figura 2.16: Representación de aplicar la transición $\delta(p, a) = r$.

Después de $k-2$ aplicaciones de la función de transición, se alcanza la configuración rz_{k-2}, como lo muestra la figura 2.17:

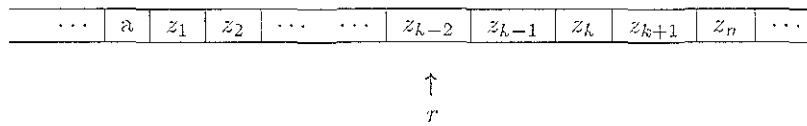


Figura 2.17: Representación de aplicar la transición $k-2$ veces.

Finalmente aplicando la función de transición $\delta(r, z_{k-2}) = q$ se obtiene la configuración qy tal que $y = z_{k-1}z_k \cdots z_{k+1} \cdots z_n$, como se muestra en la figura 2.18:

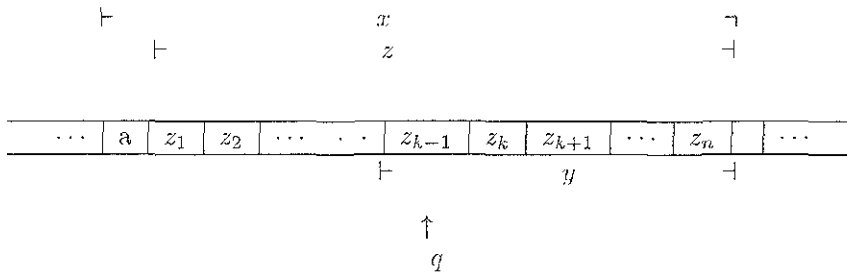


Figura 2.18: Representación de la configuración qy.

Sea el AFD $M = (Q, \Sigma, \delta, q_0, F)$ y sean px y qy dos configuraciones de M.

- Se escribe $px \vdash^{+} qy$ si $px \vdash^{k} qy$, para algún $k \geq 1$.
- Se escribe $px \vdash^{*} qy$ si $px \vdash^{k} qy$, ya sea $px = qy$, o $px \vdash^{+} qy$.

tal que:

$+$	implica un número finito de pasos no nulo
$*$	implica cero o más pasos.

Sea el AFD $M = (Q, \Sigma, \delta, q_0, F)$, se dice que una cadena $x \in \Sigma^*$ es *aceptada* o *reconocida* por M si ocurre que $sx \vdash^{*} f$, para algún $f \in F$. Sea la cadena x tal que $x = x_1x_2 \cdots x_n$, se siguen los siguientes pasos.

- *Etapa de Inicio.* colocar bien la cadena en la cinta, se coloca la cabeza de la unidad de control en la primera celda como en la figura 2.19. Si por error existe un blanco en medio de la cadena se reconoce sólo una subcadena.

Figura 2.19: Etapa de inicio.

- *Etapa de reconocimiento:* en un número finito de ciclos básicos de operación (tal vez cero) $\vdash^*$ como se muestra en la figura 2.20:

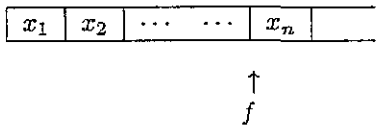


Figura 2.20: Etapa de reconocimiento.

La máquina es abstracta, no está implantada en un dispositivo físico, por lo tanto se da el supuesto que la cinta está vacía y es seminfinita.

Si la función de transición de un AFD es *total* el AFD es *completo* de otra forma es *incompleto.*

En general puede decirse que una configuración de M es una cadena de $Q\Sigma^*$ que cumple lo siguiente:

- $px \vdash qy$ si $x = ay$ tal que $a \in \Sigma$ y $\delta(p, a) = q$.
- $px \vdash^k qy$ $(k \geq 1)$
  - si $k = 1$ y $px \vdash qy$.
  - si $k > 1$ $px \vdash rz$, $rz \vdash^{k-1} qy$.
- $px \vdash^+ qy$ si $px \vdash^k qy$ para algún $k \geq 1$.
- $px \vdash^* qy$ si $px = qy$ o $px \vdash^+ qy$.

El *lenguaje* reconocido por M es el conjunto de palabras reconocidas por el autómata y puede representarse con:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* f, f \in F\}$$

La familia de los lenguajes reconocidos por los AFD se representa por:

$$L_{AFD} \subset \Sigma^*.$$

Dos AFD M_1 y M_2 se dice que son *equivalentes* si y sólo si $L_1 = L_2$

No se comparan por elementos sino por la operación esto es, reconocen al mismo lenguaje.

2.2 Diagramas de Transición.

Para vizualizar y representar un autómata finito puede usarse un diagrama de transición, en el cual los vértices representan los *estados* y las flechas o arcos representan las *transiciones.* Las etiquetas de los vértices representan los nombres de los estados, mientras que los nombres de las flechas, representan valores

actuales de los símbolos de entrada.

El diagrama debe estar completamente definido, debe haber por lo menos un arco para cada símbolo del alfabeto, de lo contrario la máquina puede llegar a ese estado y enfrentarse a una situación donde no pueda aplicarse ninguna transición.

Un *diagrama de transición* es un grafo dirigido con etiquetas que permite reconocer a las cadenas que pertenecen a un lenguaje dado.

Un diagrama de transición es *determinista* si cumple con:

- cada estado del diagrama tiene sólo un arco que sale para cada símbolo del alfabeto.
- está completamente definido.

Esto es, un problema de autómatas finitos puede pasar a ser un problema de teoría de gráficas.

Puede establecerse entonces la relación entre un *grafo* y un *autómata finito* como lo muestra la tabla 2.3.

Grafo	**Autómata Finito**
Conjunto Finito de Nodos	Estados
Conjunto Finito de Aristas Dirigidas	Transiciones
Conjunto de Etiquetas	Símbolos de Entrada

Tabla 2.3. Relación entre un Grafo y un Autómata Finito.

Se dice que se tiene un estado de *reconocimiento exitoso* si se establece un camino que conecta al nodo inicial con algún nodo final.

Por lo tanto un problema de *pertenencia* puede ser transformado en uno de *caminos*.

Cada lenguaje puede ser representado por al menos un grafo. Puede haber grafos que no tengan nodos finales y por lo tanto no habrá reconocimiento, para que exista reconocimiento deben existir nodos finales conectados con otros nodos.

Más formalmente, si $M = (Q, \Sigma, \delta, q_0, F)$, es un autómata finito determinístico, entonces su diagrama de transición asociado G_m tiene exactamente $\mid Q \mid$ vértices, cada uno está etiquetado con un $q_i \in Q$ diferente.

Para cada regla de transición:

$$\delta = (q_i, a) = q_j.$$

el grafo tiene una flecha (q_i, q_j) etiquetada con a. El vértice asociado con q_0 es llamado *vértice inicial* y los vértices $q_j \in F$ son llamados *vértices finales*.

Una *tabla de transiciones* es un arreglo (o matriz) bidimensional cuyos elementos proporcionan el resumen del diagrama de transiciones correspondiente. Los renglones representan los estados, las columnas los símbolos y la intersección entre ambos representa el estado resultante.

Para el siguiente ejemplo. el *grafo D* está representado en la figura 2.21:

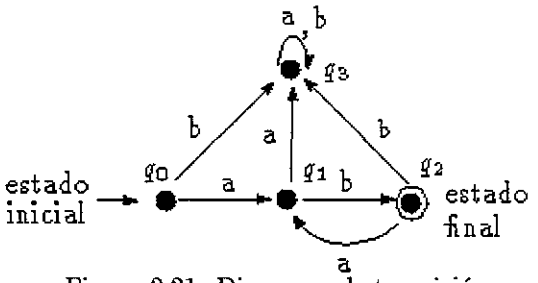


Figura 2.21: Diagrama de transición.

La *matriz de transiciones* a partitr del grafo D se representa por la tabla 2.4:

estado	a	b
q_0	q_1	q_3
q_1	q_3	q_2
q_2	q_1	q_3
q_3	q_3	q_3

Tabla 2.4: Tabla de transición para el grafo D.

Es decir. la tabla de transición es una liga entre la máquina y el grafo. donde la primera construye un sistema dinámico cuyos cambios son transiciones en el tiempo y el segundo. indica relaciones y establece caminos como una secuencia de nodos y aristas.
El proceso de reconocimiento de una cadena es el siguiente:

1. Procesar la cadena $x = aaaaa$

   - Haciendo un análisis a través del diagrama de transición se obtiene:

$$camino \begin{cases} nodos: & q_0q_1q_3q_3q_3q_3 \\ aristas: & (q_0, q_1, a), (q_1, q_3, a), (q_3, q_3, a), (q_1, q_3, a), (q_1, q_3, a) \end{cases}$$

   tal que la notación para las aristas es:

$$(q_i, q_j, x), (q_j, q_k, y)$$

significa que se va del estado:

- q_i al estado q_j leyendo el símbolo x y
- a partir del estado q_j se va al estado q_k leyendo una y.

Por lo tanto, la cadena no se reconoce porque no hay camino al nodo terminal desde el nodo inicial.

- Haciendo un análisis a través del autómata finito se obtiene:

  - Configuración inicial (figura 2.22).

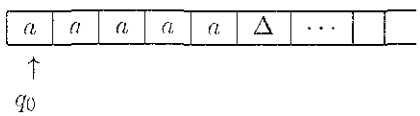


Figura 2.22. Configuración inicial en autómata finito

  - y las transiciones a partir de la tabla se muestran en la figura 2 23:

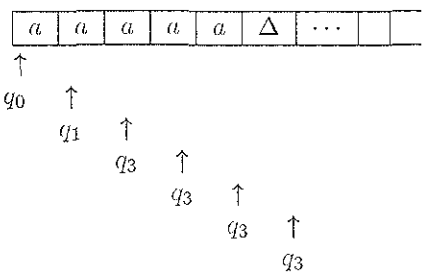


Figura 2.23: Representación de las transiciones del autómata.

Se registra el Δ y el proceso se detiene

Se sabe si la cadena pertenece o no al lenguaje, por el contenido de la unidad de control al encontrar Δ si ésta tiene un estado terminal, *pertenece* y, *no pertenece* si tiene cualquier otro estado. En este caso la unidad de control tiene al estado q_3 el cual no es un estado final. Por lo tanto se dice que *la cadena no pertenece al lenguaje.*

Como puede observarse el contenido de la cinta nunca se altera porque sólo existen dos operaciones, *movimiento* y *lectura.*

El proceso anterior se ilustra gráficamente como:

- *Etapa de Inicio*: Se coloca la cadena a analizar, cada símbolo en una celda de la cinta. Se pone la unidad de control apuntando a la primer celda registrando ésta, al estado inicial como se muestra en la figura 2.24:

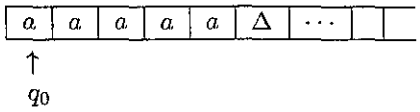


Figura 2.24: Representación de la etapa de inicio.

- *Cómputo*: Se aplica la función de transición hasta alcanzar el símbolo de fin de cadena (Δ).

⋮

- *Etapa Final*: Se hace el examen de pertenencia del autómata representado en la figura 2.25:

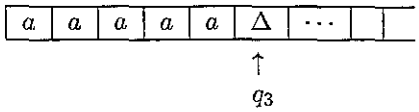


Figura 2.25: Representación de la etapa final.

Δ es un símbolo de la cinta. No debe confundirse con un símbolo del alfabeto o con la cadena vacía. Aunque se puede dejar vacía la celda y tiene el mismo efecto, pero por claridad se pone Δ.

2. Otro ejemplo es procesar la cadena $x = ababab$, representada en la figura 2.26:

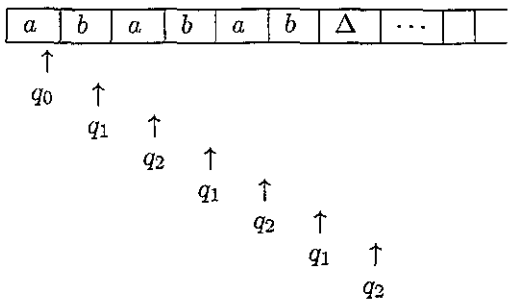


Figura 2.26: Procesamiento de la cadena $x = ababab$

El proceso se detiene en la configuración final leyendo el símbolo Δ con el estado q_2 y como éste es un estado de aceptación, la cadena es reconocida y se interpreta que la cadena x, pertenece al lenguaje $x \in L(D)$.

Siempre se debe analizar el espacio vacío o un delimitador esto es, la cadena no debe contener espacios en blanco.

En general la operación de la máquina se puede representar como en la tabla 2.27:

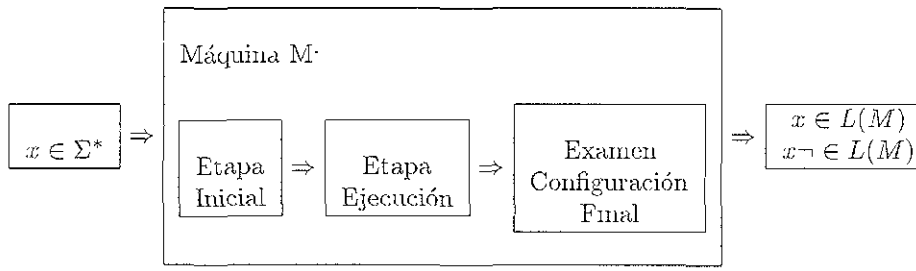


Figura 2.27: Operación de una máquina M.

Un autómata finito $M = (Q, \Sigma, \delta, q_0, F)$, se dice que es *incompleto* si $\delta : Q \times \Sigma \to Q$ es una función parcial.

Por ejemplo sea el autómata finito definido por:

$$\text{AFD} \begin{cases} Q = \{q_0, q_1\} \\ q_0 \quad \text{estado inicial} \\ F = \{q_0, q_1\} = Q \\ \Sigma = \{a, b\} \end{cases}$$

δ se define por la tabla.

estados	a	b
q_0	q_1	q_0
q_1	$-$	q_0

Tabla 2.5: Tabla de transición para el autómata finito.

donde el símbolo $-$ representa el hecho que no hay transición. Sean los siguientes casos:

- Para $x = q_0abab$ analizando el comportamiento de la máquina se obtiene:

  $q_0abab \vdash q_1bab \vdash q_0ab \vdash q_1b \vdash q_0b \vdash q_0$

  La cadena se acepta porque q_0 es un estado final $q_0 \in F$.

- Para $x = q_0ababa$

  $q_0ababa \vdash q_1baba \vdash q_0aba \vdash q_1ba \vdash q_0a \vdash q_1$

  La cadena se acepta porque q_1 es un estado final $q_1 \in F$.

- Para $x = abaab$

  $q_0abaab \vdash q_1baab \vdash q_0aab \vdash q_1ab \vdash$ no existe transición.

El lenguaje que se reconoce es:

$$L(M) = \{w \in \{a, b*\} \mid \text{la secuencia no presenta más de dos a's seguidas}\}$$

Sea N el autómata que se desprende de completar el autómata M, entonces N tiene la siguiente tabla de transición, creando un nuevo estado para la completitud.

estados	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

Tabla 2.6: Tabla de transición para el autómata finito.

De tal forma que

$$L(N) = L(M).$$

esto es que los autómatas son *equivalentes*.

No se aumenta la potencia de un AFD incompleto al completarlo.

2.2.1 Gramáticas Regulares

Otra forma de describir a los lenguajes regulares, es a través de ciertas gramáticas simples. que sean capaces de generarlos.

Una gramática es *regular*, si

- el lado izquierdo de cualquier regla de reescritura consiste en un sólo símbolo no terminal
- y el lado derecho es:
  - un sólo símbolo terminal,
  - un símbolo terminal seguido por un símbolo no terminal,
  - un símbolo no terminal seguido por un símbolo terminal,
  - un solo símbolo no terminal o la cadena vacía.

Más formalmente se pueden definir como gramáticas lineales por la derecha o lineales por la izquierda.

Recordando el concepto de gramática presentado en el capítulo 1, una *gramática* es una cuádrupla

$$G = (V, T, S, P)$$

donde

V	es un conjunto finito de objetos llamados *variables*,
T	un conjunto finito de objetos llamados símbolos *terminales*.
$S \in V$	es un símbolo inicial llamado variable de *inicio*,
P	es un conjunto finito de *producciones*.

Se puede definir una gramática $G = (V, T, S, P)$ *lineal por la derecha* si todas las producciones son de la forma:

$$A \to xB, \qquad A \to x$$

donde $A, B \in V$ y $x \in T^*$.

Una gramática es *lineal por la izquierda* si todas las producciones son de la forma:

$$A \to Bx, \qquad A \to x$$

Una gramática es *regular* si está en cualquiera de las dos formas. La importancia de las gramáticas regulares reside en que los lenguajes generados por ellas son exactamente aquellos que reconocen los autómatas finitos.

A continuación se presentan varios ejemplos de gramáticas

1 La gramática $G_1 = (\{S\}, \{a\ b\}, S, P_1)$ con P_1

$$S \to abS \mid a$$

es una gramática lineal por la derecha. Algunas derivaciones pueden ser:

1. $S \to abS \to ababS \to abababS \to abababa.$
2. $S \to abS \to aba.$
3. $S \to a.$

El lenguaje generado por la gramática G_1 es el lenguaje regular :

$$L(G_1) = \{(ab)^*a\}$$

2. $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ con P_2

$$\begin{aligned} S &\to S_1ab. \\ S_1 &\to S_1ab \mid S_2. \\ S_2 &\to a \end{aligned}$$

es una gramática lineal por la izquierda. Algunas derivaciones pueden ser:

1. $S \to S_1ab \to S_1abab \to S_1ababab \to S_2ababab \to aababab$
2. $S \to S_1ab \to S_1abab \to S_2ababab \to aababab.$
3. $S \to S_1ab \to S_2ab \to aab.$

El lenguaje generado es el lenguaje regular:

$$L(G_2) = \{a(ab)^*\}.$$

3. La gramática $G = (\{S, A, B\}, \{a, b\}, S, P)$ con P

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aB \mid \lambda \\ B &\rightarrow Ab \end{aligned}$$

no es una gramática regular, aunque las producciones son lineales por la derecha y por la izquierda la gramática en sí. no lo es. Ya que todas las derivaciones deben ser por la derecha o bien todas deben ser derivaciones por la izquierda.

Aunque ésta es una gramática lineal. se puede ver que una gramática regular es siempre lineal, pero no toda gramática lineal es regular.

Un lenguaje L es regular si y sólo si existe una gramática regular G que genere al lenguaje.

2.2.2 Propiedades de los Lenguajes Regulares

Las propiedades de clausura de las familias de lenguajes bajo diferentes operaciones resultan ser muy interesantes. La siguiente tabla muestra las propiedades de esta familia.

Sean L_1 y L_2 lenguajes regulares entonces se cumplen las propiedades de la tabla 2.7.

Operación	Notación
Unión	$L_1 \cup L_2$
Intersección	$L_1 \cap L_2$
Concatenación	$L_1 L_2$
Complemento	${L_1}^c$
Kleen	L_1^*
Reversa	L_1^R
Morfismo	$h(L_1)$
Cociente	$\frac{L_1}{L_2}$

Tabla 2.7: Propiedades de los lenguajes regulares.

2.3 Lenguajes Independientes del Contexto

Los lenguajes regulares son útiles y se describen fácilmente, sin embargo hay lenguajes que no pueden ser descritos por los lenguajes regulares.

El ejemplo más relevante de esta limitación es el lenguaje:

$$L = \{a^n b^n \mid n \geq 0\},$$

si se sustituye el paréntesis izquierdo por la a y el derecho por la b, entonces las cadenas como:

$$((())),\ ((((()))))),\ ().$$

están en L, pero la cadena (() no pertenece al lenguaje. Ésto no puede ser detectado por los autómatas finitos ya que éstos son incapaces de recordar

Este es un reconocimiento muy importante en los lenguajes de programación que describe una estructura anidada de paréntesis. Por esta causa se extiende la familia de los lenguajes a los *lenguajes independientes del contexto*, la cual tiene una gran importancia dentro de la teoría de lenguajes formales.

Las gramáticas regulares están restringidas en dos formas:

- el lado izquierdo debe ser una sola variable y
- el lado derecho debe tener una forma especial,

para hacer más poderosas a las gramáticas deben hacerse menos estrictas estas condiciones.

Las gramáticas independientes del contexto, a diferencia de las gramáticas regulares, no tienen restricciones respecto a la forma de su lado derecho de sus reglas de reescritura, aunque el lado izquierdo de cada regla debe seguir siendo un sólo no terminal.

Estas grámaticas son llamadas *independientes del contexto* porque el lado izquierdo de cada regla de reescritura contiene un sólo no terminal, entonces la regla se puede aplicar sin importar el contexto donde se encuentre este no terminal. Para hacer una diferencia marcada, la siguiente regla de producción no es independiente del contexto:

$$xNy \rightarrow xzy$$

puesto que para poder reescribir el símbolo no terminal N, éste depende del contexto, debe estar entre una x y una y.

Una gramática $G = (V, T, S, P)$ es *independiente del contexto* si todas las producciones en P son de la forma:

$$A \rightarrow x,$$

tal que $A \in V$ y $x \in (VUT)^*$.

Esta última notación indica, como se mencionó anteriormente, que el lado derecho de la regla de producción puede ser cualquier combinación de símbolos terminales con no terminales.

Un lenguaje L es un *lenguaje independiente del contexto* si y sólo si hay una gramática independiente del contexto G tal que $L = L(G)$.

Ahora se muestran ejemplos de gramáticas independientes del contexto.

1. La gramática G = $(\{S\}, \{a, b\}, S, P)$, con producciones

$$\begin{aligned} S &\to aSa, \\ S &\to bSb, \\ S &\to \lambda, \end{aligned}$$

es una gramática independiente del contexto. Una derivación cualquiera de esta gramática es:

$$S \to aSa, \to aaSaa \to aabSbaa \to aabbaa$$

entonces el lenguaje generado por la gramática es:

$$L(G) = \{ww^R \mid w \in \{a, b\}^*\}$$

que es un lenguaje independiente del contexto pero no es regular.

2. La gramática G, con producciones

$$\begin{aligned} S &\to abB, \\ A &\to aaBb, \\ B &\to bbAa, \\ S &\to \lambda. \end{aligned}$$

es independiente del contexto y el lenguaje generado es:

$$L(G) = \{ab(bbaa)^n bba(ba)^n \mid n \geq 0\}$$

Entonces puede decirse que cada gramática regular es independiente del contexto, pero no toda gramática independiente del contexto es regular.

2.3.1 Derivaciones por la Izquierda y por la Derecha

Las gramáticas independientes del contexto generan cadenas como cualquier otra gramática, pero como éstas permiten más de un no terminal del lado derecho, hay ambigüedad al querer reescribir una regla con más de un símbolo no terminal.

La siguiente gramática independiente del contexto puede generar ambigüedad en la reescritura de los símbolos no terminales M y N.

$$\begin{aligned} S &\to zMNz, \\ M &\to aMa, \\ M &\to z, \\ N &\to bNb, \\ N &\to z. \end{aligned}$$

Para generar la cadena $x = zazabzbz$

1. Escogiendo M o el símbolo no terminal más a la izquierda.
   $S \Rightarrow zMNz \Rightarrow zaMaNz \Rightarrow zazaNz \Rightarrow zazabNbz \Rightarrow zazabzbz.$

2. Escogiendo N. o el símbolo no terminal más a la derecha.
   $S \Rightarrow zMNz \Rightarrow zMbNbz \Rightarrow zMbzbz \Rightarrow zaMabzbz \Rightarrow zazabzbz$

La derivación de **1.** se llama *derivación por la izquierda* y la **2.** se llama *derivación por la derecha.* El orden en el que se apliquen las reglas de reescritura no afecta la generación de la cadena en cualquier gramática independiente del contexto. Si una cadena puede generarse a partir de una derivación , entonces puede generarse por una derivación por la izquierda. Cada una de estas derivaciones tiene el nombre de *formas sentenciales.*

En el siguiente ejemplo se muestra como se cambia el orden de las formas sentenciales obteniendo, efectivamente, la misma cadena. Sea la gramática definida por:

$$\begin{aligned} S &\to AB \\ A &\to aaA \\ A &\to \lambda \\ B &\to Bb \\ B &\to \lambda \end{aligned}$$

En las siguientes dos derivaciones, se puede observar que además de producir la misma cadena, se usan exactamente las mismas producciones.

1. $S \Rightarrow^1 AB \Rightarrow^2 aaAB \Rightarrow^3 aaB \Rightarrow^4 aaBb \Rightarrow^5 aab$

2 $S \Rightarrow^1 AB \Rightarrow^4 ABb \Rightarrow^2 aaABb \Rightarrow^5 aaAb \Rightarrow^3 aab$

2.3.2 Árboles de Derivación

Otra forma de mostrar las derivaciones aparte de la notación anterior, sin importar el orden en el que se apliquen las reglas son los árboles de derivación.

Los *árboles de derivación* son árboles tal que sus nodos representan símbolos terminales y no terminales de la gramática de la siguiente forma:

- el nodo raíz es el símbolo de inicio de la gramática.
- los hijos de cada nodo no terminal son los símbolos que reemplazan a ese no terminal en la derivación.
- ningún símbolo terminal puede ser un nodo interior del árbol,
- ni ningún símbolo no terminal puede ser una hoja.

Para la gramática G. con producciones:

$$\begin{aligned} S &\to aAB. \\ A &\to bBb \\ B &\to A \mid \lambda \end{aligned}$$

- La cadena $abBbB$ es una forma sentencial de G representada por el árbol de la figura 2.28:

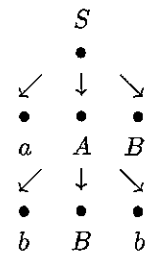


Figura 2.28: Árbol para la forma sentencial $abBbB$.

- La cadena $abbbb$ es llamada una sentencia de $L(G)$.

Los árboles de derivación son una forma explícita de describir las derivaciones, como los diagramas de transición en los autómatas finitos.

Sea una gramática G independiente del contexto, para cada:

$$w \in L(G),$$

existe un árbol de derivación de G que produce w y cada producción de cualquier árbol derivación está en $L(G)$.

Aunque los árboles de derivación muestran de forma clara las producciones que se usan, no muestran el orden de aplicación.

Para una gramática $G = (V, T, S, P)$ indpendiente del contexto, un *árbol ordenado* es un árbol de derivación para G si y sólo si cumple las siguientes propiedades:

1. La raíz se etiqueta con S.
2. Cada hoja tiene un nivel de $T \cup \{\lambda\}$.
3. Cada vétice interior (vértice que no es hoja) tiene un nivel de V.
4. Si un vértice tiene etiqueta $A \in V$, y sus hijos están etiquetados (de izquierda a derecha) $a_1, a_2, \ldots, a_n$ entonces P debe contener una producción de la forma:

$$A \to a_1, a_2, \ldots, a_n.$$

5. Una hoja etiquetada con λ no tiene hermanos esto es, un vértice etiquetado con λ no pude tener hijos.

Un árbol que cumple con las propiedades 3,4 y 5, pero no con 1 necesariamente, la propiedad 2 se reemplaza por:

2a. Cada hoja tiene un nivel de $V \cup T \cup \{\lambda\}$.

y es llamado un *árbol de derivación parcial.*

La cadena de símbolos que se obtiene de leer las hojas del árbol de derivación de izquierda a derecha (sin contar las hojas etiquetadas con λ), se llama una producción.
Una *producción* es la cadena de terminales que se encuentra cuando el árbol es recorrido en profundidad siempre iniciando con la rama más a la izquierda.
Por ejemplo, sea la siguiente gramática independiente del contexto:

$$\begin{aligned} E &\rightarrow \quad +TE' \\ E' &\rightarrow \quad TE' \mid \lambda \\ T &\rightarrow \quad FT' \\ T' &\rightarrow \quad *FT' \mid \lambda \\ F &\rightarrow \quad b \mid (E) \end{aligned}$$

Leyendo las hojas del árbol de derivación de izquierda a derecha sin tomar en cuenta las hojas etiquetadas con λ, se obtiene la producción $+b$. Tal que los vértices interiores están encerrados en un rectángulo y las hojas están encerradas en un círculo. la raíz está denotada con la palabra *raíz*, como lo muestra la figura 2.29:

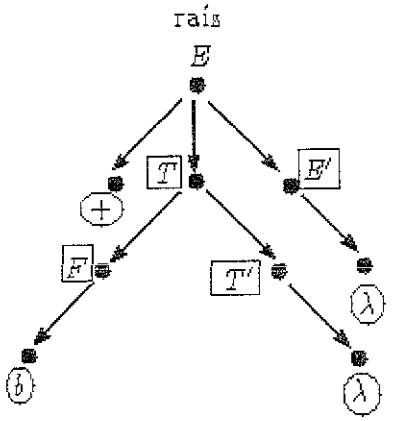


Figura 2.29: Árbol de derivación para la cadena $+b$.

Se presenta una *ambigüedad* cuando para alguna cadena $w \in L(G)$ hay más de un árbol de derivación.

Una gramática independiente del contexto es *ambigua* si existe alguna cadena $w \in L(G)$ que tiene al menos dos árboles de derivación diferentes. Alternativamente, la ambigüedad implica la existencia de dos o más derivaciones por la derecha o por la izquierda.

Por ejemplo, la gramática con producciones:

$$S \rightarrow \quad aSb \mid SS \mid \lambda$$

es ambigüa ya que la cadena *aabb* tiene los dos árboles de derivación siguientes en la figura 2.30:

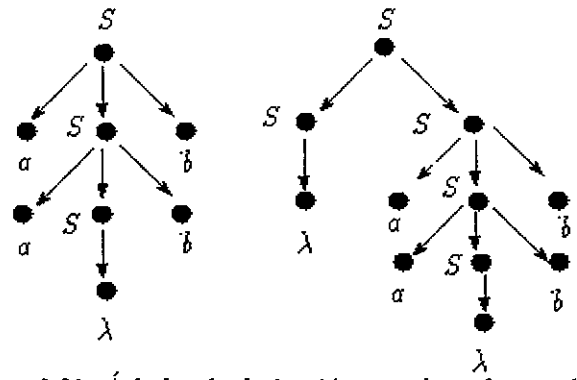


Figura 2.30: Árboles de derivación para la cadena *aabb*.

2.3.3 Métodos para Transformar Gramáticas

Uno de los principales problemas en el estudio de gramáticas y lenguajes es la presencia de la cadena vacía. Aunque juega un papel importante en muchos teoremas y demostraciones, es preferible eliminarla, considerando sólo los lenguajes que no contienen a la cadena vacía.

Sea L cualquier lenguaje independiente del contexto y sea $G = (V.T, S, P)$ una gramática independiente del contexto para:

$$L - \{\lambda\}$$

entonces la gramática que se obtiene al agregar a V (tal que V es el conjunto de símbolos variables de G) la nueva variable S_0, haciendo S_0 el símbolo de inicio y agregando también a P las producciones:

$$S_0 \rightarrow S \mid \lambda$$

genera L. Por lo tanto cualquier conclusión no trivial que se pueda hacer para:

$$L - \{\lambda\}.$$

se puede hacer para L. Además dada cualquier gramática independiente del contexto G, hay un método para obtener G', tal que:

$$L(G') = L(G) - \{\lambda\}$$

Consecuentemente, para todos los propósitos prácticos no hay diferencia entre los lenguajes independientes del contexto que incluyen a λ y los que no.

Además de lo relacionado con el símbolo λ, se puede llevar a las gramáticas independientes del contexto a formas simplificadas equivalentes que permiten un fácil manejo de éstas.

1. **Regla de Sustitución Útil.**
Una regla útil para simplficar gramáticas consiste en eliminar producciones *indeseables*: el proceso no necesariamente resulta en una reducción real del número de reglas.

Sea una producción del tipo:

$$A \to x_1 B x_2$$

y otra del tipo:

$$B \to y_1 \mid y_2 \mid \cdots \mid y_n$$

tal que A y B son dos variables diferentes y éstas son todo el conjunto de producciones en P, entonces se puede obtener la gramática simplificada $G' = (V, T, S, P')$ tal que la producción $A \to x_1 B x_2$ puede ser eliminada de la gramática, si se pone en su lugar un conjunto de producciones en las cuales B es reemplazada por todas las cadenas que deriva en un paso de la siguiente forma:

$$A \to x_1 y_1 x_2 \mid x_1 y_2 x_2 \mid \cdots \mid x_1 y_n x_2.$$

Entonces.

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

Por ejemplo. sea $G = (\{A, B\}, \{a, b, c\}, A, P)$ con producciones.

$$\begin{aligned} A &\to \quad a \mid aaA \mid abBc. \\ B &\to \quad aabA \mid b \end{aligned}$$

utilizando la sustitución mencionada para la variable B, se genera la gramática G' con producciones:

$$\begin{aligned} A &\to \quad a \mid aaA \mid abaabAc \mid abbc. \\ B &\to \quad aabA \mid b \end{aligned}$$

la nueva gramática G' es equivalente a G. la cadena $aaabbc$ tiene la derivación

$$A \Rightarrow aaA \Rightarrow aaabBc \Rightarrow aaabc$$

en G. y la derivación correspondiente:

$$A \Rightarrow aaA \Rightarrow aaabc$$

en G'. En este caso la variable B y sus producciones asociadas permanecen en la gramática aunque no tengan un papel importante en ninguna derivación.

2. **Eliminación de Producciones Inútiles.**

Es muy importante eliminar de las gramáticas las producciones que nunca forman parte de una derivación, por ejemplo la gramática:

$$\begin{aligned} S &\rightarrow aSb \mid \lambda \mid A, \\ A &\rightarrow aA \end{aligned}$$

de la producción $S \rightarrow A$, se nota que A no puede ser transformada en una cadena terminal. Eliminando esta producción se tiene el mismo lenguaje.

Una variable es *inútil* si no hay forma de alcanzar una cadena terminal desde ella. Otra razón por la que una variable se denomina inútil es la siguiente.

Por ejemplo sea la siguiente gramática:

$$\begin{aligned} S &\rightarrow A, \\ A &\rightarrow aA \mid \lambda, \\ B &\rightarrow bA \end{aligned}$$

la variable B es inútil y también lo es la producción:

$$B \rightarrow bA$$

aunque B puede derivar en una cadena terminal, no hay forma de alcanzar :

$$S \Rightarrow^* xBy$$

Con este ejemplo se ilustran dos razones por las cuales una variable es inútil, porque no puede ser alcanzada desde el símbolo inicial o porque no pueden derivar una cadena terminal.

Para la gramática:

$$G = (V, T, S, P),$$

donde:

$$V = \{S, A, B, C\},$$

y

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

con P:

$$\begin{aligned} S &\to aS \mid A \mid C. \\ A &\to a. \\ B &\to aa. \\ C &\to aCb. \end{aligned}$$

Primero, se identifican las variables que conducen a una cadena terminal.

$$A \to a \qquad y \qquad B \to aa$$

las variables A y B, pertenecen a este conjunto. Para S, se puede ver que:

$$S \Rightarrow A \Rightarrow a.$$

Sin embargo para la variable C, no sucede lo mismo, entonces es una *variable inútil*. Removiendo C y su producción correspondiente se llega a la gramática G_1 con.

$$V_1 = \{S, A, B\},$$

los terminales:

$$T_1 = \{a\}.$$

y producciones:

$$\begin{aligned} S &\to aS \mid A. \\ A &\to a. \\ B &\to aa. \end{aligned}$$

Para eliminar las variables que no pueden ser alcanzadas desde el símbolo inicial, se dibuja la *gráfica de dependencia* para las variables. La gráfica de dependencia es una forma de visualizar las relaciones que son complejas para las gramáticas independientes del contexto. los nodos representan las variables y las flechas entre los nodos representan que dichas variables están relacionadas por una producción de la gramática.

Por ejemplo, sean los nodos C y D, hay una flecha de C a D si existe la producción:

$$C \to xDy$$

La figura 2.31 muestra la gráfica de dependencia para V_1:

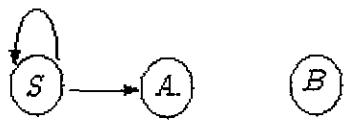


Figura 2.31: Gráfica de dependencia.

Una variables es *útil* si hay una ruta desde el vértice etiquetado con S al vértice de la variable. Por lo tanto eliminando B y las producciones relacionadas con B, se llega a la gramática:

$$G' = (V', T', S, P'),$$

donde:

$$V' = \{S, A\},$$

y

$$T' = \{a\}.$$

con las producciones:

$$\begin{aligned} S &\to aS \mid A, \\ A &\to a \end{aligned}$$

3. **Eliminación de Producciones** λ.

La mayoría de las veces no es deseable tener del lado derecho de la producción a la cadena vacía. Cualquier producción de una gramática independiente del contexto de la forma:

$$A \to \lambda$$

es llamada una *producción-*λ. Cualquier variable A, para la cual es posible la derivación:

$$A \Rightarrow^* \lambda$$

es llamada *anulable.*

Una gramática puede generar un lenguaje que no contenga λ. aunque tenga producciones λ o variables anulables. En estos casos. se pueden eliminar las transiciones λ.

Por ejemplo, la gramática:

$$\begin{aligned} S &\to aS_1b. \\ S_1 &\to aS_1b \mid \lambda. \end{aligned}$$

genera el lenguaje *libre de* λ:

$$\{a^n b^n \ , n \geq 1\}.$$

La *producción-*λ, $S_1 \to \lambda$ puede ser eliminada después de poner dos producciones nuevas, que se obtienen por sustituir λ por S_1, cuando ésta aparece en el lado derecho, haciendo esta transformación se obtiene la gramática:

$$\begin{aligned} S &\to aS_1b \mid ab. \\ S_1 &\to aS_1b \mid ab. \end{aligned}$$

es fácil ver que esta nueva gramática genera el mismo lenguaje, que la original.

Sea la siguiente gramática independiente del contexto, definida por:

$$\begin{aligned} S &\to ABaC. \\ A &\to BC \mid \lambda. \\ B &\to b \mid \lambda. \\ C &\to D \mid \lambda. \\ D &\to d. \end{aligned}$$

Para eliminar las transiciones λ, se deben considerar los siguientes puntos:

(a) Encontrar el conjunto V_n de todas las variables anulables de G, usando los siguientes pasos:
   i. Para todas las producciones $A \to \lambda$, poner A en V_n.

(b) Repetir el siguiente paso para toda variable en V_n.
   Para todas las producciones:

$$B \to A_1A_2 \cdots A_n.$$

   donde $A_1, A_2, \cdots A_n \in V_n$. Poner B en V_n.

   Después de haber encontrado el conjunto V_n, se construye el nuevo conjunto de producciones P'. Para ésto, se consideran todas las producciones de P que tienen la forma:

$$A \to x_1x_2 \cdots x_m, m \geq 1$$

   donde $\forall x_i \in V \cup T$ Para cada producción de P de esta forma, se pone en P' esta producción y todas las que se generan cuando se reemplazan las variables anulables con λ en todas las combinaciones posibles. Por ejemplo, si x_i y x_j son anulables, entonces habrá una producción en P' con x_i que reemplaza a λ, una con x_j que reemplaza a λ y otra con x_i, x_j que reemplaza a λ. Hay una excepción, si x_i es anulable, la producción.

$$A \to \lambda$$

no está en P'.

Entonces para la eliminación de λ, primero se encuentran las variables anulables, para la gramática de ejemplo éstas son $V_N = \{A, B, C\}$, porque son de la forma:

$$A \to \lambda$$

Después, para la construcción del segundo paso, se toma la primer variable del conjunto V_N, en este caso se sabe que la variable A es de la forma:

$$A \to BC \mid \lambda$$

entonces se aplica la sustitución de λ en todas las producciones donde aparezca la variable A. En este ejemplo sólo aparece en el lado derecho del símbolo S, esto implica tener la nueva transformación:

$$\begin{array}{ll} S \to & ABaC \mid BaC, \\ A \to & BC, \\ B \to & b \mid \lambda, \\ C \to & D \mid \lambda, \\ D \to & d. \end{array}$$

Haciendo el mismo procedimiento para la siguiente variable del conjunto V_N (la variable B), se obtiene:

$$\begin{array}{ll} S \to & ABaC \mid BaC \mid AaC \mid aC. \\ A \to & BC \mid C, \\ B \to & b, \\ C \to & D \mid \lambda, \\ D \to & d. \end{array}$$

Finalmente para la variable C se obtiene la gramática transformada, sin la variable λ:

$$\begin{array}{ll} S \to & ABaC \mid BaC \mid AaC \mid aC \mid ABa \mid Ba \mid Aa \mid a, \\ A \to & BC \mid C \mid B. \\ B \to & b. \\ C \to & D. \\ D \to & d. \end{array}$$

Por último se debe notar que uno de los usos más importantes de la teoría de lenguajes formales es en la definición de lenguajes y en la construcción de sus intérpretes o compiladores.

Capítulo 3

Sistemas de Reescritura

Los orígenes de los sistemas de reescritura se remontan a la primera parte del siglo veinte, cuando Axel Thue, propuso el siguiente problema:

Supóngase que se tiene un conjunto de objetos y un conjunto de transfromaciones (reglas), que cuando se aplican a estos objetos producen objetos en el mismo conjunto. Dados dos objetos x y y en el conjunto. Se quiere saber si puede x ser transformado en y, o hay un tercer objeto z tal que x y y pueden ser transformados en z.

Este problema fue conocido como el *problema de la palabra*. Thue estableció algunos resultados preliminares de las cadenas de símbolos que pueden ser extendidos a objetos combinatoriales más complejos como gráficas o árboles. Además intentó desarrollar un *cálculo*, para decidir el problema de la palabra esto es, un conjunto de procedimientos o algoritmos que pudieran ser aplicados para obtener la respuesta correcta. Es decir, un algoritmo general para resolver el problema de la palabra en una variedad de conjuntos diferentes.

Aproximadamente en los mismos años Dehn, estuvo trabajando con lo que resultaría ser el reforzamiento de la teoría de grupos combinatoria e introdujo el problema de la palabra para grupos. Aparentemente el trabajo de Thue, estuvo abandonado por muchos años, hasta que éste fue retomado por lógicos para proporcionar definiciones de las nociones de *algoritmo* y *procedimiento efectivo*. Esto fue completamente natural, ya que Thue quería saber si el problema de la palabra era *decidible*.

En la mitad de 1950 y principios de 1960, los sistemas de reescritura fueron estudiados en lingüística matemática y en teoría de lenguajes formales ya que fueron muy útiles en modelos matemáticos para gramáticas estructuradas por frases. Estas gramáticas fueron ampliamente estudiadas dentro del contexto del problema de traducción de lenguajes naturales.

3.1 Sistemas de Reescritura

Las matemáticas aparecen en todos lados, algunas veces se trata de determinar si una identidad es consecuencia lógica de algunos axiomas. Otras veces, se trata de encontrar soluciones a algún tipo de ecuaciones. Este tipo de razonamientos puede tener muchas aplicaciones en la computación, incluyendo computación algebraica simbólica, prueba de teoremas de autómatas, verificación y especificación de programas, lenguajes de programación de alto nivel, etc.

Los *sistemas de reescritura* son ecuaciones dirigidas en las que se van reemplazando subtérminos repetidamente de una fórmula, hasta que la forma más simple es alcanzada.

Las *ecuaciones dirigidas* son las reglas de reescritura, que se utilizan para reemplazar unos términos por otros, en la dirección indicada.

Los sistemas de reescritura pueden ser de diferentes tipos por ejemplo,

- Reescritura de cadenas.
- Reescritura de términos.
- Reescritura de gráficas.
- Reescritura condicional.
- Reescritura de prioridad.
- Reescritura restringida.
- Reescritura paralela.
- Reescritura secuencial.
- Reescritura infinita.

Generalmente dentro de la computación, los sistemas de reescritura se estudian en las siguientes dos direcciones:

1. Lenguajes de Programación.
    - Semánticas.
    - Lógica Combinatoria.
    - Cálculo Lambda.
    - Lenguajes de Programación Lógicos y Funcionales.
2. Deducción Automatizada.
    - Computación Algebraica y Simbólica.
    - Procedimientos de Decisión.
    - Pruebas de Teoremas Basadas en Reescritura.
    - Unificación.

Como un formalismo, los sistemas de reescritura tienen una gran potencia en las máquinas de Turing y éstas, pueden ser vistas como algoritmos de Markov no determinísticos, sobre términos o sobre cadenas.

3.2 Dispositivos Generativos y de Reconocimiento

Un *sistema de reescritura* es un par ordenado:

$$SR = (\Sigma, F)$$

donde.

Σ es un alfabeto y
F es un conjunto finito de pares ordenados de palabras sobre Σ.

Los elementos:

$$(P, Q) \in F.$$

son llamados *reglas de reescritura* o *producciones* y se denotan con:

$$P \rightarrow Q$$

Una palabra P sobre Σ *genera directamente* una palabra Q en símbolos:

$$P \Rightarrow Q$$

si y sólo si existen palabras:

$$P'.P_1.P''.Q_1$$

tal que:

$$P = P'P_1P''. \quad Q = P'Q_1P''$$

y

$$P_1 \rightarrow Q_1 \in F$$

Una palabra P *genera a* Q.

$$P \Rightarrow^* Q$$

si y sólo si existe una secuencia finita de palabras sobre Σ.

$$P_0.P_1.\ldots.P_k. \qquad k \geq 0$$

donde:

$$P_0 = P. \quad P_k = Q$$

y P_i genera directamente a

$$P_{i+1} \qquad \forall \qquad 0 \leq i \leq k-1.$$

Sea $W(\Sigma)$ el conjunto de todas las palabras sobre el alfabeto Σ entonces.

- $\Rightarrow$ es una relación binaria sobre el conjunto W(Σ) y

- $\Rightarrow^*$ es la cerradura transitiva reflexiva de $\Rightarrow$.

Un sistema de reescritura puede convertirse en un *dispositivo generativo*, si se especifica un conjunto de axiomas X de $\mathrm{W}(\Sigma)$, y se considera el lenguaje:

$$L_g(X) = \{Q | P \Rightarrow^* Q, \quad P \in X\}$$

como el *conjunto generado* por el sistema de reescritura a parir del conjunto de axiomas X.

Además un sistema de reescritura puede ser un *dispositivo de reconocimiento* y el lenguaje:

$$L_g(X) = \{P | P \Rightarrow^* Q, \quad Q \in X\}$$

es el *lenguaje reconocido o aceptado* por el sistema de reescritura y el conjunto de axiomas X.

La mayoría de las veces, el conjunto de axiomas X, consiste de un sólo elemento y si no, su estructura es bastante simple.

Si se consideran los dos lenguajes anteriores modificándolos de tal forma que se dividide el alfabeto en dos alfabetos:

- terminales Σ_T
- y los no terminales Σ_N

y se da un enfoque en el subconjunto de las palabras *terminales* y *no terminales* esto es, el *alfabeto de los terminales* y *alfabeto de los no terminales* y tomando en cuenta que sólo las palabras que pertenencen al alfabeto de terminales son las que forman los lenguajes L_g.

Un *algoritmo normal de Markov* es un sistema de reescritura:

$$SR = (\Sigma, F),$$

con elementos de F en un orden lineal:

$$P_1 \to Q_1$$
$$\vdots$$
$$P_k \to Q_k$$

Además si se da un subconjunto F_1 del conjunto de producciones, los elementos de F_1. son llamadas producciones finales y se denotan por $P \to Q$.

En cada paso del sistema de reescritura la primera producción aplicable debe ser la más a la izquierda.

Este algoritmo posee una propiedad que no poseen todos los sistemas de reescritura en general es *monogénico*. ya que para cualquier palabra P hay a lo más una palabra que puede ser producida a partir de P en un solo paso.

Se dice que P genera a Q directamente. $P \Rightarrow Q$ si se cumplen cada una de las siguientes condiciones:

1. Existe i. $1 \leq i \leq k$. y palabras P' y P'' tal que $P = P'P_iP''$ y $Q = P'Q_iP''$.
2. Ninguna de las palabras P_j, tal que $j < i$. es subpalabra de P.

3. P_i ocurre como una subpalabra de $P'P_i$ solo una vez.

4. $P_i \to Q_i$ no es un elemento de F_1, y una de las palabras $P_1, \ldots P_k$ es una subpalabra de Q.

El proceso de reescritura termina con una aplicación de una producción final o cuando ninguna de las producciones es aplicable.

- $P \Rightarrow Q$ si y sólo si 1. y 3. se satisfacen, pero la condición 4. no se cumple.
- $P \Rightarrow^* Q$ si y sólo si hay una secuencia finita $P = R_0, R_1, \ldots R_u, Q$ tal que.

$$R_j \Rightarrow R_{j+1} \quad 0 \leq j \leq u-1$$

y

$$R_u \Rightarrow Q$$

o de lo contrario

$$P = Q$$

y ninguna de las palabras $P_1, \ldots P_k$ es una subpalabra de P.

Entonces, para cualquier P, hay a lo más una palabra Q, tal que $P \Rightarrow^* Q$. Si tal Q existe el algoritmo normal se confunde con P y traduce P en Q. De otra forma el algoritmo se encicla en P.

Por ejemplo, sea el alfabeto $\{a, b\}$ y las producciones:

$$\{a \to \lambda, b \to \lambda, \lambda \to aba\}$$

traduce cada palabra sobre el alfabeto $\{a, b\}$ en la palabra aba. Todas las letras a y b, se eliminan con las primeras dos producciones y luego λ se reescribe como aba.

El algoritmo normal con el alfabeto {a,X,Y,#} y las producciones:

$$\{Ya \to aY, Xa \to aYX, X \to \lambda, a\# \to \#X\}.$$

$$\{\#a \to \#, \# \to \lambda, Y \to a\}$$

traduce cada palabra:

$$a^i \# a^j, i, j \geq 0$$

en la palabra a^j, el algoritmo normal multiplica dos números.

El algoritmo normal con el alfabeto $\{a, X, Y, Z, \#\}$ y las producciones:

$$\{aX \to Xa, a\#a \to X\#, a\# \to \#Y, Y \to a\},$$

$$\{X \to Z, Z \to a\# \to \lambda\}$$

traduce cada palabra en la palabra a^k, tal que k es el máximo común divisor entre i, j.

3.3 Aplicaciones de los Sistemas de Reescritura

Los tipos de sistemas de reescritura que han sido más ampliamente estudiados, son los que operan con cadenas de caracteres. Aunque también se ha hecho una importante aportación con sistemas de reescritura en otras formas de operación, por ejemplo Wolfram, planteó sistemas de reescritura en arrays para simular autómatas celulares.

La primera definición formal de estos sistemas, fue dada por Thue, sin embargo en los años 50's, el trabajo de Chomsky sobre gramáticas formales originó un gran interés en los sistemas de reescritura con cadenas. El concepto de reescritura se aplicó para describir características sintácticas de los lenguajes naturales. Posteriormente Backus y Naur introdujeron una notación basada en la reescritura para describir formalmente el lenguaje de programación ALGOL-60. Además, se encontró la equivalencia de la forma Backus-Naur con las gramáticas independientes del contexto de Chomsky y con esto surgió la aplicación de la *sintáxis* y la *gramática* a la ciencia de la computación.

Con los avances de la teoría de lenguajes formales, surgió la idea de usar cadenas de caracteres para describir gráficas. Los primeros objetivos fueron reconocer letras escritas a mano o cromosomas, a través de la codificación de sus imágenes y analizando las cadenas resultantes. Después se encontró la relación entre las imágenes y las clases de lenguajes en una manera formal, sin embargo los lenguajes que estaban relacionados con las imágenes más simples resultaron ser *sensibles al contexto*, aunque hubo muchas aplicaciones útiles de la teoría de lenguajes formales en el área de *lenguajes independientes del contexto*.

A continuación se muestran algunas de las aplicaciones más interesantes de los sistemas de reescritura al campo de la computación científica, describiendo en la mayoría de los casos el fenómeno de crecimiento con mecanismos de reescritura. La técnica básicamente consiste en iniciar con una configuración o contorno (en el caso de imágenes) y reemplazar sucesivamente las partes usando el conjunto de reglas de reescritura.

3.3.1 Fractales

Uno de los primeros ejemplos de objetos matemáticos definidos por un proceso de reescritura es la *curva de Koch*, la cual es obtenida recursivamente en escala de una curva original y utilizando el patrón de reducción en la mitad de cada tercio de segmento de la figura. Estos objetos han sido ampliamente estudiados por Benoit Mandelbrot para definir el concepto de *fractales*.

Un *fractal* es un contorno geométrico fragmentado que puede ser subdividido en partes, cada una de las cuales es, (al menos aproximadamente) una copia reducida de la figura completa, como lo muestra la figura 3.1

La geometría fractal fue introducida por Mandelbrot para describir fenómenos naturales complejos tal como perímetros de islas, montañas, galáxias, nubes, etc. Además demostró que los objetos construídos con esta geometría llamados objetos fractales pueden representar patrones de estrellas en el cielo, redes de fluidos en nuestro organismo, etc. La geometría fractal es un nuevo campo en las matemáticas y parece proporcionar una nueva forma mucho más natural para el estudio del universo.

Figura 3.1: Fractales.

Se puede describir la curva de Koch en términos de reglas de reescritura de la siguiente forma, se comienza con dos contornos un *iniciador* y un *generador*. Este último se divide en N segmentos iguales de longitud r. Entonces en cada etapa de la construcción se comienza con un segmento de los resultantes de la división y este se reemplaza con una copia del generador reducida al tamaño del intervalo a ser reemplazado.Supóngase el iniciador de la figura 3.2:

Figura 3.2: Iniciador.

y el generador de la figura 3.3:

Figura 3.3: Generador.

depués del primer proceso de reescritura se obtiene la figura 3.4:

Figura 3.4: Primera reescritura.

y aplicando recursivamente el proceso de reescritura se obtiene el fractal de la figura 3.5:

Figura 3.5: Reescritura sucesiva.

3.3.2 Autómatas Celulares

El popular *juego de la vida* de John H. Conway, es una trivial y famosa aplicación de la teoría de autómatas celulares. Este, es sin duda el mejor ejemplo para explicar la teoría de los autómatas celulares.

Esta teoría fue inicialmente desarrollada por John Von Neumann, para probar la capacidad de autorreproducción de una máquina. Sin embargo, esto fue tan sólo un componente de lo que vendría a ser una teoría compleja de autómatas celulares.

Un *autómata celular* es un espacio o conjunto de células con las siguientes propiedades:

- las células están distribuidas regularmente en un *espacio N dimensional*.
- cada célula está, en toda generación, en un estado de *un conjunto finito de estados*.
- una *configuración* de un autómata celular está definida como el conjunto de todas las células de esa generación.
- un *estado* de una célula en una generación depende exclusivamente de los estados de sus células vecinas.
- una *función de transición* define el estado de una célula a partir de los estados de sus vecinos de la transición anterior.

Una característica importante de los autómatas celulares es que todos los estados de transición ocurren a la vez en pasos discretos de tiempo, formando evoluciones sucesivas como modelos de evolución.

El objetivo del juego de la vida consiste en encontrar una configuración inicial que evolucione a muchas generaciones (no debe desaparecer rápidamente). Los autómatas celulares pueden servir como modelos explícitos para una amplia variedad de procesos biológicos, químicos y físicos como: formación de patrones biológicos, turbulencia de fluidos, etc.

El conjunto de estados es:

$$\Sigma = \{0, 1\} \begin{cases} 0 & \text{representa una célula muerta.} \\ 1 & \text{representa una célula viva.} \end{cases}$$

Además se deben de cumplir las condiciones para que una célula nasca, sobreviva o muera.

1. *Nacimiento.* Una célula muerta en la generación k llega a vivir en la siguiente generación $(k+1)$ si tiene exactamente tres vecinos vivos en la generación k.
2. *Muerte por sobrepoblación.* Si una célula vive en la generación k y tiene cuatro o más vecinos vivos, esta célula debe morir en la generación siguiente.
3. *Muerte por aislamiento.* Si una célula vive en la generación k y tiene ningún o un vecino vivo en esa misma generación, esta célula debe morir en la siguiente generación.
4. *Sobrevivencia.* Una célula que vive en la generación k, deberá permanecer viva en la siguiente generación si tiene dos o tres vecinos vivos en esa misma generación.

A continuación se muestra en la figura 3.6, un ejemplo de *life* donde las células permanecen constantes:

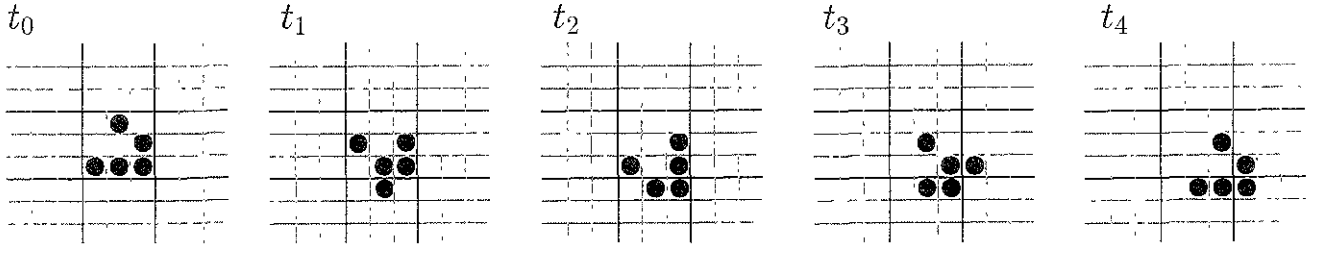


Figura 3.6: Reescritura en life de permanencia.

La figura 3.7 muestra el ejemplo de *life* donde las células mueren:

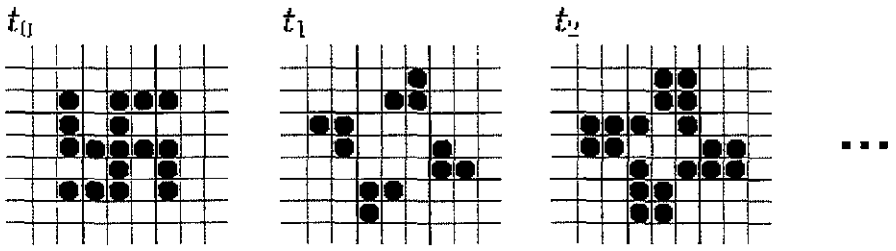


Figura 3.7: Reescritura en life de muerte.

3.3.3 Máquinas de Turing

La clase de autómatas que se conoce como *Máquinas de Turing* fue propuesta por Alan M. Turing en 1936. La idea básica de Turing fue estudiar los procesos algorítmicos utilizando un modelo computacional. En ese mismo año, Emil Post presentó un enfoque similar y más tarde se mostró que ambas estrategias son equivalentes en cuanto a poder computacional. Las máquinas de Turing pueden ser vistas como una versión generalizada de los autómatas finitos y de los de pila. Además se asemejan a los autómatas finitos en que constan de un mecanismo de control y un flujo de entrada que se concibe como una cinta; la diferencia es que las máquinas de Turing pueden mover sus cabezas de lectura hacia adelante y hacia atrás y pueden leer o escribir en la cinta. Estas características aumentan en gran medida la capacidad de las máquinas.

Las acciones específicas que puede realizar una máquina de Turing consisten en operaciones de escritura y movimiento.

La *operación de escritura* consiste en reemplazar un símbolo y luego cambiar a un nuevo estado (el cual puede ser el mismo donde se encontraba antes).

La *operación de movimiento* comprende mover la cabeza una celda a la derecha o a la izquierda y luego pasar a un nuevo estado (que, una vez más puede ser igual al de partida).

La acción que se ejecutará en un momento determinado dependerá del símbolo (actual) que está en la celda visible en ese momento para la cabeza (la celda actual), así como el estado actual del mecanismo de control de la máquina.

Si se representa con:

- Γ el conjunto de símbolos de una máquina de Turing,
- con S el conjunto de todos los estados, y
- con S', el conjunto de estado que no son el de parada,

entonces es posible representar las transiciones de la máquina por medio de una función, llamada *función de transición* de la máquina de la forma:

$$\delta : (S' x T) \rightarrow (\Gamma \cup \{L, R\})$$

donde se supone que los símbolos L y R no pertenecen a Γ.

La semántica de esta representación funcional es la siguiente:

1. $\delta(p, x) = (q, y)$ significa *si el estado actual es p y el símbolo actual es x, reemplazar la x con el símbolo y y pasar a estado q.*
2. $\delta(p, x) = (q, L)$ significa *si el estado actual es p y el símbolo actual es x, mover la cabeza una celda a la izquierda y pasar a estado q.*
3. $\delta(p, x) = (q, R)$ significa *si el estado actual es p y el símbolo actual es x, mover la cabeza una celda a la derecha y pasar a estado q.*

Al describir con una función las transiciones de una máquina de Turing, la máquina es *determinista*. Existe, una y sólo una, transición asociada a cada par estado − símbolo donde el estado no es el de detención.

Durante la operación normal, una máquina de Turing ejecuta transiciones repetidamente hasta llegar al estado de parada. Esto quiere decir que en ciertas condiciones es posible que nunca se detengan los cálculos de una máquina de Turing ya que su programa interno puede quedar atrapado en un ciclo sin fin

Es útil representar la máquina de Turing visualmente. El mecanismo de control de la máquina está representado por un rectángulo que indica el estado actual de la máquina. debajo de este rectángulo se encuentra la cinta de la máquina; la posición de la cabeza de la máquina está representada por la flecha como lo muestra la figura 3.8:

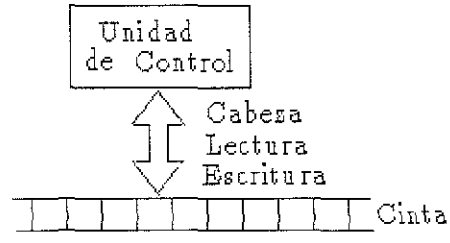


Figura 3.8: Maquina de Turing.

Un modelo formal para un procedimiento efectivo deberá poseer ciertas propiedades.

1. Cada procedimiento deberá poderse describir de manera finita.
2. El procedimiento deberá consistir en pasos discretos, cada uno de los cuales puede llevarse a cabo de manera mecánica.

De manera formal, una *máquina de Turing* se representa por:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$$

donde:

- Q es el conjunto finito de estados.
- Γ es el conjunto de símbolos de cinta admisibles.
- $\#$ símbolo de Γ, es el espacio en blanco.

- $\Sigma \subset \Gamma$ que no incluye a #, es el conjunto de símbolos de entrada.
- δ es la función de transición, es una transformación de $Qx\Gamma$ a $Qx\Gamma x\{L, R\}$.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Por ejemplo, sea un elemento de la función de transición:

$$\delta(q_0, a) = (q_1, d, L)$$

donde:

- la máquina de Turing se encuentra en el estado q_0 leyendo el símbolo a,
- el proceso de reescritura consiste en sustituir el símbolo a por d y pasar al estado q_1.
- el símbolo L. (*left*) significa que la cabeza de lectura hace un movimiento a la izquierda.

La figura 3.9 ilustra el proceso de reescritura:

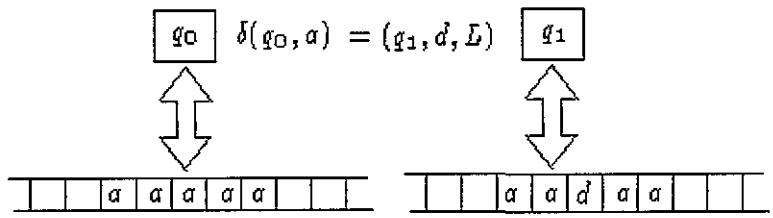


Figura 3.9: Reescritura en máquinas de Turing.

3.3.4 Sistemas de Reescritura de Lindenmayer

Como ya se mencionó anteriormente, los sistemas de Lindenmayer. hacen representaciones de imágenes utilizando cadenas de caracteres. Son sistemas de reescritura en paralelo, que tienen como configuración inicial un axioma y utilizando las reglas de reescritura comienzan a generar cadenas que simulan procesos evolutivos de organismos celulares.

Por ejemplo, sea el sistema de Lindenmayer descrito por:

- *alfabeto:*

$$\{a, b, c, d, e, f, g, h, i, j, k\}.$$

- *producciones:*

$$\left\{\begin{array}{ll} \mathrm{a} & \rightarrow bc. \\ \mathrm{b} & \rightarrow kd. \\ \mathrm{c} & \rightarrow ek. \\ \mathrm{d} & \rightarrow bb. \\ \mathrm{e} & \rightarrow cf. \\ \mathrm{f} & \rightarrow ii. \\ \mathrm{g} & \rightarrow bc. \\ \mathrm{h} & \rightarrow de. \\ \mathrm{i} & \rightarrow k. \\ \mathrm{k} & \rightarrow k \end{array}\right\}$$

- *axioma:*

$$a$$

Este sistema produce la siguiente secuencia:

a
bc
kdek
kgbcfk
khikdekihk
kdekkgbcfkkdek
kgbcfkkhikdekihkkgbcfk
khikdekihkkdekkgbcfkkdekkhikdekihk

Las células en el estado k corresponden a las porciones que no tienen crecimiento en el margen de la hoja.

Cada una de las cadenas generadas por una gramática L, tiene una interpretación geométrica, es decir, símbolo a símbolo se va interpretándo geométricamente para generar la imagen del proceso. Esta graficación ha sido realizada en la mayoría de los casos con la interpretación de la *geometría de la tortuga.*

La *geometría de la tortuga*, generalmente recibe los comandos:

- Avanzar hacia adelante n pasos.
- Avanzar hacia atrás n pasos.
- Girar a la derecha α grados.
- Girar a la izquierda α grados.

de tal forma que cada símbolo en la cadena generada tiene asociado un comando o una combinación de estos comandos.
Por ejemplo, el símbolo b, obedece al comando *avanzar hacia adelante y girar a la derecha*, y cada símbolo tiene su propio comando asociado como avanzar a la derecha- girar a la izquierda, solo girar a la izquierda, etc. De esta forma se pueden dibujar las correspondientes a las cadenas:

1. *kdekkgbcfkkdek*, esta cadena interpretada gráficamente se muestra en la figura 3.10:

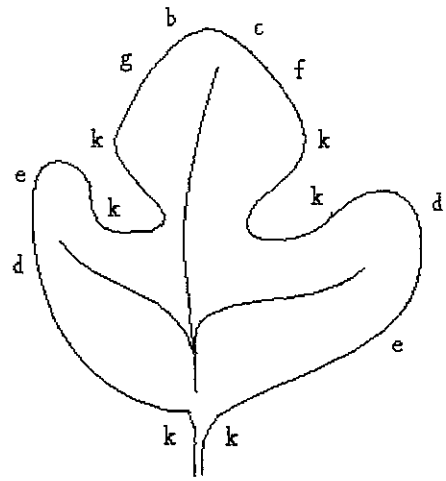


Figura 3.10: Cadena *kdekkgbcfkkdek*.

2. *kgbcfkkhikdekihkkgbcfk*, de igual forma esta cadena se representa en la figura 3.11 gráficamente:

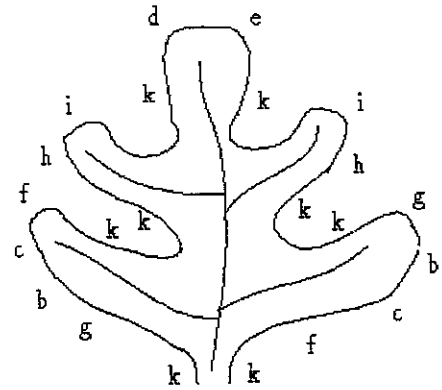


Figura 3.11: Cadena *kgbcfkkhikdekihkkgbcfk*.

Capítulo 4

Fundamentos Teóricos de los Sistemas de Lindenmayer

El tipo más simple de los sistemas de Lindenmayer, son los sistemas llamados *L0D*, los cuales son determinísticos e independientes del contexto.

Aunque matemáticamente son muy simples, estos sistemas proporcionan una clara visión de las ideas y técnicas básicas de los sistemas de Lindenmayer y del paralelismo en la reescritura en general.

Los primeros ejemplos de los sistemas de Lindenmayer que se utilizaron para modelar desarrollos biológicos fueron los *sistemas L0D*.

4.1 Sistemas de Lindenmayer Independientes del Contexto (L0)

Un *sistema L0* es una terna $G = (\Sigma, h, w_0)$ donde:

- Σ es un alfabeto,
- h es una sustitución finita en Σ (conjunto finito de producciones) y
- w_0 (axioma) es una palabra sobre Σ.

Una *producción* es un par ordenado (a, χ) y se escribe:

$$a \rightarrow \chi$$

La letra a es llamada *predecesor* y χ es el *sucesor* de la producción.

Para cualquier letra $a \in \Sigma$, hay al menos una palabra $\chi \in \Sigma^*$ tal que $a \rightarrow \chi$, donde Σ^*, representa el conjunto de palabras sobre Σ.

Si una regla de producción no está especificada explícitamente para un predecesor $a \in \Sigma$ se asume que la *producción identidad*

$$a \rightarrow a$$

pertenece al conjunto de producciones del sistema.

Si $\mu = a_1 \dots a_m$ es una palabra arbitraria sobre Σ, entonces la palabra $\nu = \chi_1 \dots \chi_m \in \Sigma^*$ es *derivada directamente* de μ y se denota por:

$$\mu \Rightarrow \nu \quad \text{siempre que} \quad a_i \rightarrow \chi_i \quad \forall i = 1, \dots m$$

Una palabra ν tiene una *derivación de longitud n si existe una secuencia de desarrollo* de palabras generadas a partir del axioma w:

$$\mu_0, \mu_1, \dots, \mu_n \quad \text{tal que} \quad \mu_0 = w, \mu_n = \nu \quad \text{y} \quad \mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n.$$

El sistema G va generando palabras cada vez que se aplican las reglas de producción a partir del *axioma* w_0. el conjunto de estas palabras, se conoce como el *lenguaje generado por* G y se denota de la siguiente forma:

$$L(G) = \{w_0\} \cup h(w_0) \cup h(h(w_0)) \cup \dots = \bigcup_{i \geq 0} h^i(w_0)$$

El mecanismo de los procesos de reescritura de los sistemas de Lindenmayer se puede ilustrar con el siguiente ejemplo utilizado para simular el desarrollo de un filamento multicelular encontrado en la bacteria azul-verde *Anabaena catenula.*

- Los símbolos a y b representan los estados citológicos de las células.
- Los subíndices i (izquierda) y d (derecha) indican la polaridad de la célula, especificando las posiciones en las cuales, las células hijas serán producidas.

La modelación del desarrollo está representado por las siguientes reglas de reescritura:

$$\begin{array}{lll} p_1: & a_d \rightarrow & a_i b_d, \\ p_2: & a_i \rightarrow & b_i a_d, \\ sp_3: & b_d \rightarrow & a_d, \\ p_4: & b_i \rightarrow & a_i, \end{array}$$

Iniciando con la célula a_d (como axioma), el sistema de Lindenmayer genera la siguiente secuencia de palabras:

$$\begin{array}{l} a_d \\ a_i b_d \\ b_i a_d a_d \\ a_i a_i b_d a_i b_d \\ b_i a_d b_i a_d a_d b_i a_d a_d \\ \vdots \end{array}$$

Visto en un microscopio, las *células se observan como rectángulos* de varias longitudes. Las células de tipo a. son más grandes que las células de tipo b. Esto puede ser representado gráficamente por rectángulos, los grandes para a y los más pequenõs para b. Además las polaridades pueden ser representadas por flechas internas en los rectángulos. Por ejemplo, el axioma a_d, se representa por un rectángulo grande con una flecha interna cuya dirección va a la derecha. La siguiente palabra generada es $a_i b_d$ y es representada por un rectángulo grande con flecha interna a la izquierda y un rectángulo menor con flecha a la derecha. Siguiendo este razonamiento puede interpretarse gráficamente cualquier palabra generada.

La figura 4.1 muestra el desarrollo del sistema, para la palabra $b_l a_d b_l a_d a_d b_l a_d a_d$:

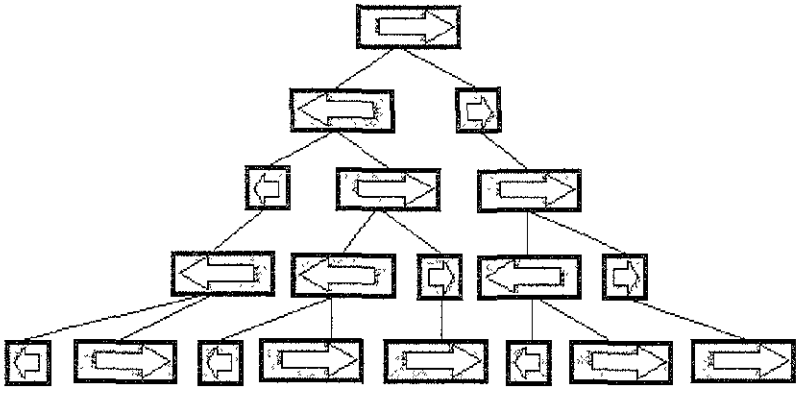


Figura 4.1: Desarrollo del alga *Anabaena Catenula*.

4.2 Sistemas de Lindenmayer Propagados (L0P)

Un *sistema L0* $G = (\Sigma, h, w_0)$ es llamado *propagado* si no tiene ninguna *producción de eliminación* que tiene la forma:

$$a \to \lambda$$

en su conjunto de producciones. De otra forma el sistema es llamado *no propagado*.

4.3 Sistemas de Lindenmayer Determinísticos (L0D)

Un *sistema L0* $G = (\Sigma, h, w_0)$ es *determinístico* si y sólo si para cada $a \in \Sigma$ existe exactamente una α en Σ^* tal que:

$$a \to \alpha$$

Es decir, para todo símbolo del alfabeto hay exactamente una regla de producción definida. De otra forma el sistema es llamado *no determinístico*.

1. El siguiente sistema de Lindenmayer G es *propagado* pero no es *determinístico*.

$$G = \{\{a\}, \{a \to a, a \to a^3\}, a\}$$

analizando el conjunto de producciones de este sistema:

$$a \to a, a \to a^3$$

se nota que hay más de una producción especificada para el símbolo a (*no determinismo*) y que la producción de eliminación $a \rightarrow \lambda$ no pertenence al conjunto de producciones (*propagación*).

En general, un sistema puede ser definido simplemente por el listado de las reglas de producción especificando el axioma, (el alfabeto Σ y la sustitución h pueden ser leídos de las producciones). Los sistemas de Lindenmayer pueden definirse también de esta manera. es decir, por su lista de producciones.

Por ejemplo, el sistema L G anterior puede definirse como:

$$\{a \rightarrow a, \qquad a \rightarrow a^3, \text{ con axioma } a.\}$$

2. El siguiente sistema de Lindenmayer, es *determinístico* y *propagado.*

   - *alfabeto:*

   $$\{S, a, b, c, d, e, f, g, h, i, j, k, m, 0, 1, 2\}$$

   - *reglas de producción:*

   $$\begin{array}{ll} S \rightarrow & ab, \\ a \rightarrow & dg, \\ b \rightarrow & e0, \\ c \rightarrow & 22, \\ d \rightarrow & 0e, \\ e \rightarrow & cf, \\ f \rightarrow & 1c, \\ g \rightarrow & hb, \\ h \rightarrow & di, \\ i \rightarrow & jk, \\ j \rightarrow & m1, \\ k \rightarrow & c0, \\ m \rightarrow & 0c, \\ 0 \rightarrow & 0, \\ 1 \rightarrow & 1, \\ 2 \rightarrow & 2 \end{array}$$

   - *axioma*

   $$S$$

Los primeros ocho pasos de derivación a partir de S son:

(a) $S \Rightarrow ab$

(b) $ab \Rightarrow dge0$

(c) $dge0 \Rightarrow 0ehbcf0$

(d) $0ehbcf0 \Rightarrow 0cfdie0221c0$

(e) $0cfdie0221c0 \Rightarrow 0221c0ejkcf0221220$

(f) $0221c0cjkcf0221220 \Rightarrow 0221220cfm1c0221c0221220$

(g) $0221220cfm1c0221c0221220 \Rightarrow 0221220221c0c1220221220221220$

(h) $0221220221c0c1220221220221220 \Rightarrow 022122022122022122022122022122020.$

En esta serie de derivaciones, puede observarse que la última cadena deriva en si misma, entonces $L(G)$, consiste en las nueve palabras listadas.

$$L(G) = \{\ S, ab, dgc0.0ehbcf0.0cfdie0221c0, 0221c0ejkcf0221220, 0221220cfm1c0221c0221220, 0221220221c0c1220221220221220, 022122022122022122022122022122020\ \}.$$

Este es un ejemplo del desarrollo de una célula en el contorno de una hoja. De hecho, los tipos de célula 0 1 2 corresponden al tipo de células que definen el contorno de una hoja de la siguiente manera:

- 0- células que están en las *muescas* entre los lóbulos.
- 1- células en la punta de los lóbulos.
- 2- células del contorno donde no hay división.

Con esta interpretación, la última cadena representa el contorno de una hoja como lo muestra la figura 4.2:

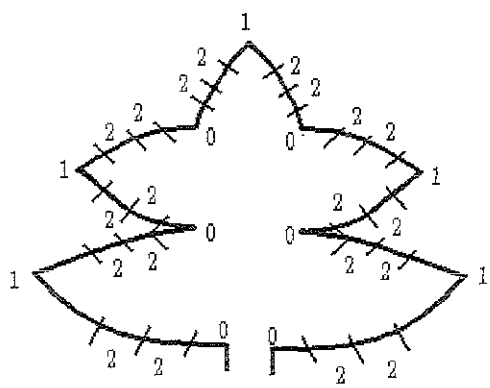


Figura 4.2: Desarrollo de un sistema de Lindenmayer *L0DP*.

Aplicando recursivamente este sistema puede obtenerse una figura más compleja como la que se muestra a continuación en la figura 4.3:

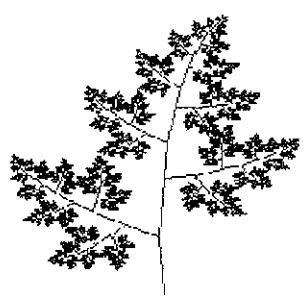


Figura 4.3: Aplicación recursiva del sistema de Lindenmayer.

4.4 Sistemas de Lindenmayer Unarios (L0U)

Un *sistema 0L* $G = (\Sigma, h, w_0)$ es *unario* si y sólo si:

$$\text{card}\Sigma = 1.$$

donde *card*Σ significa la "cardinalidad " del conjunto Σ es decir, el número total de elementos del conjunto. En este caso el alfabeto Σ consta de una sola letra.

La propiedad de estos sistemas de tener sólo un símbolo en su alfabeto, llamada *reestricción unaria* tiene ciertas ventajas:

- Los sistemas $U0L$ son muy triviales y permiten hacer una caracterización completa.
- Las cadenas, que constan de una sola letra, pueden ser puestas en correspondencia con los números naturales para un fácil manejo.
  Entonces si $\Sigma = \{a\}$, cualquier lenguaje L sobre $\{a\}$ está determinado únicamente por el conjunto S de enteros no negativos definido por
  $$S = \{n \mid a^n \in L\}.$$
  En la misma forma una cadena sobre $\{a\}$ puede ser representada por un entero y un lenguaje sobre $\{a\}$ puede ser representado por un conjunto de enteros.

Un *sistema L0U* puede ser representado como un conjunto de enteros.

Si un sistema *L0U* es determinístico, propagado o de crecimiento se agregan las letras D, P, G, (*growing*) respectivamente.

1. El siguiente sistema L llamado $OLVIDA_3$, muestra un sistema *no propagado, no determinístico* y además *unario*.

   - *alfabeto:*

     a

   - *reglas de producción:*

     $$a \to \lambda$$
     $$a \to a^2$$
     $$a \to a^5$$

   - *axioma:*

     a

   A partir del axioma a se obtiene, en una *derivación de un paso* cada una de las palabras λ, a^2, a^5, ya que el sistema es no determinístico.

   axioma **a**

λ	a^2	a^5

   Un segundo paso de derivación proporciona lo siguiente:

- de λ .
  No proporciona ninguna nueva palabra.
- de a^2.
  Sea $a^2 = a_1 a_2$ se pueden obtener las palabras:
  - a^4 de la sustitución $a_1 \to a^2,\ a_2 \to a^2$.
  - a^7 de la sustitución $a_1 \to a^2,\ a_2 \to a^5 = a_1 \to a^5,\ a_2 \to a^2$.
  - a^{10} de la sustitución $a_1 \to a^5,\ a_2 \to a^5$.
- de a^5.
  Siguiendo el procedimiento similar de a^2 se obtiene:
  $a^6, a^8, a^9, a^{11}, a^{12}, a^{13}, a^{14}, a^{15}, a^{16}, a^{17}, a^{19}, a^{20}, a^{22}, a^{25}$.

Recordando que en estos sistemas todos los símbolos de la palabra se *reescriben simultáneamente.*

Un tercer paso de derivación produce, en muchas formas diferentes las palabras que olvida el sistema.

$$a^{18}, a^{21}, a^{23}, a^{24},$$

es fácil de mostrar por inducción que.

$$L(OLVIDA_3) = \{a^i \mid i \neq 3\}$$

2. Este ejemplo representa un sistema *unario* y *determinístico* de nombre $EXPONENTE_2$ con:
   - axioma a
   - y la regla $a \to a^2$.

   las cuatro primeras palabras que genera el sistema son:

$$\begin{array}{c} a \\ a^2 \\ a^4 \\ a^8 \\ a^{16} \\ \vdots \end{array}$$

esta serie se obtiene a partir del axioma a y la regla de sustitución $a \to a^2$ de la siguiente forma:

$$\begin{array}{lll} a \to & a^2, & \Rightarrow a^2 \\ a \to & (a^2)^2, & \Rightarrow a^4 \\ a \to & (a^4)^2, & \Rightarrow a^8 \\ a \to & (a^8)^2, & \Rightarrow a^{16} \\ & \vdots & \end{array}$$

En general, el sistema genera el lenguaje formado por las palabras:

$$a^{2^i}, \qquad i \geq 0$$

4.5 Sistemas de Lindenmayer Extendidos (L0E)

Un *sistema L0* $G = (\Sigma, h, w_0, \delta)$ es *extendido* tal que:

- Σ es un *alfabeto de símbolos no terminales*,
- h es un *conjunto de reglas de producción*,
- w_0 es un *axioma*,
- δ es un *alfabeto de símbolos terminales*.

La letra E (extendido) en el nombre de un sistema de Lindenmayer significa que se permite el uso de símbolos no terminales, entonces un sistema L0E es un sistema L0, donde el alfabeto está dividido en dos partes, *símbolos terminales* y *símbolos no terminales*.

Los sistemas 0L y los L0E trabajan en la misma forma, pero sólo las palabras sobre el alfabeto terminal están en el *lenguaje* de un sistema L0E.

Un sistema L0E G puede ser visto como un sistema L0, donde se especifica un subalfabeto $\sum_T$ y el lenguaje del sistema 0L se intersecta con $\sum_T^*$ para obtener el lenguaje del sistema L0E.

El siguiente ejemplo es muy instructivo, el sistema L0E llamado *SINCRO*.

En esta notación las letras mayúsculas son símbolos no terminales y las minúsculas son símbolos terminales.

- *alfabeto de no terminales:*

$$\Sigma = A, \acute{A}, B, \acute{B}, C, \acute{C}, F$$

- *alfabeto de terminales:*

$$\delta = a, b, c,$$

- *reglas de producción:*

$$\begin{array}{lllll} A \to A\acute{A} & A \to a & \acute{A} \to \acute{A} & \acute{A} \to a & a \to F \\ B \to B\acute{B} & B \to b & \acute{B} \to \acute{B} & \acute{B} \to b & b \to F \\ C \to C\acute{C} & C \to c & \acute{C} \to \acute{C} & \acute{C} \to c & c \to F \\ & & F \to F & & \end{array}$$

- *axioma:*

$$ABC$$

Es fácil verificar que el lenguaje es:

$$L(SINCRO) = \{a^n b^n c^n \mid n \geq 1\}$$

esto es debido a que el sistema L0E es *sincronizado* en el sentido de que todos los terminales deben ser alcanzados simultáneamente.

Por otro lado el símbolo de falla F, nunca puede ser eliminado de la palabra. Este lenguaje es un ejemplo clásico en la teoría de lenguajes formales: lenguajes sensibles al contexto.

4.6 Sistemas de Lindenmayer con Tablas (L0T)

Un *sistema L0T* es una terna $G = (\Sigma, S, w_0)$, donde:

- S es un conjunto de sustituciones finitas tal que:

$$\forall \sigma \in S, \text{la terna } G = (\Sigma, \sigma, w_0), \text{ es un sistema } L0.$$

En otras palabras un sistema L0T es un sistema de Lindenmayer que tiene más de un conjunto de producciones disponibles, pero tiene el mismo alfabeto y el mismo axioma.

El *lenguaje* de un sistema L0T $L(G)$, consiste de w_0 y de todas las palabras en los lenguajes:

$$\sigma_1 \ldots \sigma_k(w_0), \text{ donde } k > 1$$

y cada σ_i pertenece a S. Además algunos σ_i pueden coincidir

Un rasgo muy característico de la reescritura en paralelo es el uso de *tablas*.

Una *tabla* es simplemente un conjunto de reglas de reescritura.

Un sistema puede tener varias tablas siempre finitas, es esencial que en cada paso del proceso de reescritura, se usen las reglas de producción de la misma tabla. Ésto causa el siguiente estado del fenómeno en la modelación del desarrollo de organismos, reales o artificiales. Puede haber diferentes condiciones en el medio ambiente (noche y día, variación de calor, variación de luz, contaminación, etc.) o diferentes fases del desarrollo, donde es importante usar diferentes reglas. La figura 4.4 muestra diferentes condiciones en el medio ambiente.

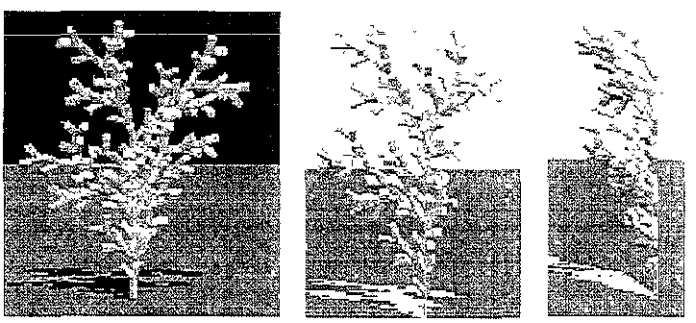


Figura 4.4: Condiciones diferentes para un sistema de Lindenmayer.

Entonces se consideran todos los conjuntos de reglas, tablas, obtenidas en esta forma. Pero se observa que las tablas no tienen sentido en la reescritura secuencial. Porque sólo se usa una regla en cada paso de derivación, esto es suficiente para considerar el conjunto total de reglas.

La letra D indica que todas las sustituciones (todas las tablas) son determinísticas. Sin embargo, de acuerdo con la definición anterior, no hay control con el uso de las tablas, (las tablas pueden ser usadas en orden arbitrario y en multitud).

Un lenguaje L0TD no es generado en una secuencia. Una secuencia definitiva resulta sólo si el orden en el uso de las tablas está especificado en forma única.

Se define un sistema L0T por la especificación del axioma en cada tabla, una tabla es un conjunto de producciones incluidas en corchetes para indicar que están juntas. No hay reestricciones, la misma producción puede aparecer en muchas tablas.

Como ejemplo se ve el sistema L0TD llamado *PAL:*

- con el *axioma:*

$$a$$

- y las dos *tablas de reescritura:*

$$T_d = [a \to b, b \to b^2, c \to a]$$
$$T_n = [a \to c. b \to ac, c \to c]$$

(Las letras d y n indican *día* y *noche* respectivamente.)

En lugar de una secuencia lineal, las derivaciones se presentan en el siguiente árbol en la figura 4.5, donde la raíz está representando al axioma:

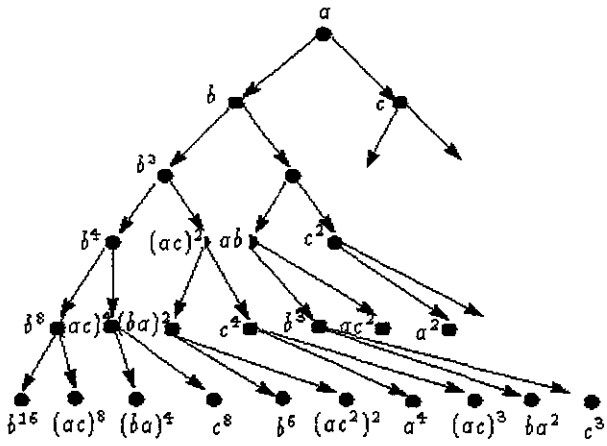


Figura 4.5: Arbol de derivación.

Cada rama indica cual de las tablas fue usada:

- un descendiente por la izquierda resulta de una aplicación de T_d,
- un descendiente por la derecha indica una aplicación de T_n.

Si no hay un descendiente marcado (como a y c en el tercer nivel) indica que éste ocurrió en un nivel anterior.

La continuación puede ser ignorada, si sólo se está interesado en determinar al lenguaje. La rama más a la izquierda del árbol contiene las potencias:

$$b^{2^i}. \forall i \geq 0.$$

Sin embargo, todas las potencias de:

$$b^i, i \geq 1,$$

ocurren en otra parte del árbol. Lo mismo sucede para las potencias:

$$a^i \text{ y } c^i, i \geq 1.$$

Primero se debe observar que:

$$\forall i \geq 0. ba^{i+1} \text{ resulta de } ba^i$$

aplicando primero T_n y luego T_d.

- De ba^i la palabra b^{i+2} resulta de aplicar T_d.
- la palabra c^{i+2} de aplicar T_n dos veces
- y la palabra a^{i-2} de c^{i+2} de T_d.

Aunque son aparentemente muy simples, los sistemas LAP tienen una rica estructura. Si se da un orden de aplicación de las tablas, resulta una única secuencia de palabras. Se pueden visualizar las dos tablas para las reglas para el día y la noche.

La alternación es el orden natural de aplicación:

$$T_d T_n T_d T_n T_d \ldots$$

(suponiendo que se comienza con T_d).

Otra posibilidad es considerar *luz del día eterna* (solo usar T_d) o *noche eterna*.

Si se escoge un *axioma arbitrario*: en vez de a, es decir la palabra no vacía w sobre $\{a, b, c\}$. (Algunas veces se usa el término *proyecto L* en lugar de sistema L, para indicar que el axioma no está especificado). Se denota $LAPDIA-NOCHE(w)$, $LAP-DIA(w)$, $LAP-NOCHE(w)$. Cada una de ellas generan una secuencia específica de palabras.

De hecho $LAP-DIA(w)$ y $LAP-NOCHE(w)$ son sistemas L0D, ya que $LAPDIA-NOCHE(w)$ puede ser visto como un sistema determinístico y propagado, o sea un L0DP.

Por ejemplo, para el axioma:

$$w = abc,$$

la secuencia es la siguiente:

- *DIA-NOCHE*:

$$abc, b^3a, (ac)^3c, (ba)^3a, (ac^2)^3c, (ba^2)^3a, (ac^3)^3c, (ba^3)^3a, (ac^4)^3c, (ba^4)^3a, \ldots$$

- *DIA*:

$$abc, b^3a, b^7, b^{14}, b^{28}, \ldots.$$

- *NOCHE*:

$$abc, cac^2, c^4, c^4, \ldots.$$

4.7 Sistemas L0 Bifurcados y de Florecencia

Un *sistema L0* $G = (\Sigma. h. w_0)$ es *bifurcado* si los símbolos:

$$[,] \in \Sigma$$

Los *sistemas L0T* pueden ser utilizados cuando ocurren cambios en el desarrollo de un organismo. La metamorfosis de un insecto o el cambio de un proceso vegetativo a un proceso reproductivo en el florecimiento de plantas pueden servir como ejemplos. La figura 4.6 muestra un proceso de florecencia.

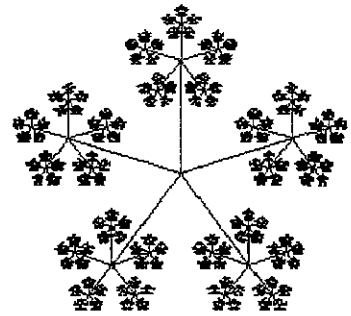


Figura 4.6: Proceso de florecimiento.

La producción del florecimiento se hace con el desarrollo de una estructura bifurcada (o ramificada), en estas ramas ocacionalmente aparecerán flores. Este proceso puede ser desarrollado por sistemas que contengan símbolos de ramificación (como son los paréntesis o corchetes). A continuación se muestra en la figura 4.7 un proceso de ramificación y florecencia.

Figura 4.7: Proceso de ramificación y florecimiento.

El mecanismo de la inducción de flores ha sido investigado ampliamente en las plantas que comienzan a florecer ya sea en el día o en la noche. Los estímulos a la inducción de florecimiento son recibidos por el maduramiento de hojas del crecimiento vegetativo de las plantas La figura 4.8 muestra un florecimiento más complejo que los anteriores.

Figura 4.8: Florecimiento.

Como una demostración de lo anterior se considera el siguiente sistema L. Este tipo de crecimiento no involucra interacciones entre las partes de la planta o con el medio ambiente.

El sistema está definido por dos tablas de produccion determinísticas *L0*. Para generar la bifurcación o estructura de ramificación las reglas de producción emplean los paréntesis.

- *alfabeto*:

$$1.2.3.4.5.6.7.8, A, B, C, (.)$$

- *tablas de reglas de producción*, descritas en la tabla 4.1:

Vegetativo	*Reproductivo*
$1 \rightarrow 21$	$1 \rightarrow A$
$2 \rightarrow 3$	$2 \rightarrow 6$
$3 \rightarrow 4$	$3 \rightarrow 8$
$4 \rightarrow 5$	$4 \rightarrow 7(1)$
$5 \rightarrow 76$	$5 \rightarrow 6$
$6 \rightarrow 7(1)$	$6 \rightarrow 7$
$7 \rightarrow 7$	$7 \rightarrow 7$
	$8 \rightarrow 7(A)$
	$A \rightarrow B$
	$B \rightarrow C$
	$C \rightarrow C$
$(\rightarrow ($	$(\rightarrow ($
$) \rightarrow)$	$) \rightarrow)$

Tabla 4.1: Reglas de producción.

- *axioma*:

El alfabeto de la primera tabla contiene como símbolos los números del 1 al 7 y los paréntesis izquierdo y derecho, la segunda tabla contiene además el número 8 y las letras *A*, *B*, *C*.

Las letras mayúsculas representan diferentes etapas en el desarrollo del florecimiento:

- La letra **A** representa botones de flores sin abrir.
- La letra **B** representa flores.
- La letra **C** representa flores que han abierto su botón y están en proceso de dar un fruto.

La figura 4.9 muestra el desarrollo *vegetativo* de las primeras etapas. Las etapas siguientes a partir de la flecha representan un desarrollo *reproductivo* hasta que la aplicación de éstas ya no producen cambios.

1
21
321
4321
54321
7654321
77(1)7654321
77(21)77(1)7654321
77(321)77(21)77(1)7654321
77(4321)77(321)77(21)77(1)7654321
77(54321)77(4321)77(321)77(21)77(1)7654321
77(7654321)77(54321)77(4321)77(321)77(21)77(1)7654321
77(77(1)7654321)77(7654321)77(54321)77(4321)77(321)77(21)77(1)7654321
77(77(21)77(1)7654321)77(77(1)7654321)77(7654321)77(54321)77(4321)77(321)77(21)77(1)7654321
→ 77(77(6A)77(A)7767(1)86A)77(77(A)7767(1)86A)77(7767(1)86A)77(67(1)86A)77(7(1)86A)77(86A)77(6A)77(A)7767(1)86A
77(77(7B)77(B)7777(A)7(A)7B)77(77(B)7777(A)7B)77(7777(A)7(A)7B)77(77(A)7(A)7B)77(7(A)7(A)7B)77(7(A)7B)77(7B)77(B)7777(A)7(A)7(b)

Figura 4.9: Desarrollo vegetativo y reproductivo.

Esta última cadena puede ser vista gráficamente de la siguiente forma, como lo muestra la figura 4.10. (las ramas se han dibujado alternando izquierda-derecha).

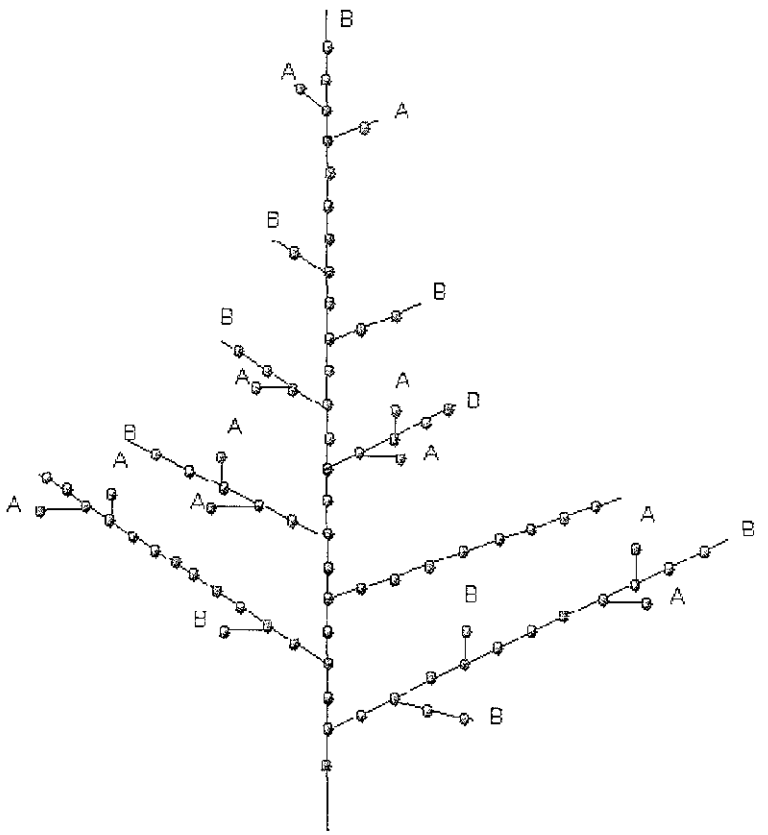


Figura 4.10: Florecimiento.

Se puede notar a través de la gráfica que la rama izquierda más cercana al tallo se repite en su parte más alta en la siguiente rama superior de ésta. Y de esta forma va repitiendo su parte más alta sucesivamente hasta terminar en una sola B Lo mismo sucede para las ramas del lado derecho.

4.8 Sistemas de Lindenmayer Estocásticos (L0S)

Un *sistema 0L* $G_\Phi = (\Sigma, P, w_0, \Phi)$ donde:

- Σ es un *alfabeto.*
- w_0 es un *axioma.*
- P es un *conjunto de producciones*
- Φ es una función de *distribución de probabilidad* tal que:

$$\phi : P \to (0, 1]$$

es un sistema *estocástico.*

La *distribución de probabilidad* ϕ es una función que mapea al conjunto de producciones en el conjunto de *probabilidades de producción.*

Para cualquier letra $a \in \Sigma$ la suma de probabilidades de todas las producciones con el *predecesor* a es igual a 1.

Una derivación estocástica en G_ϕ es de la forma:

$$\mu \Rightarrow \nu$$

si para cada *ocurrencia* de la letra a en la palabra μ la probabilidad de aplicar la producción p con el predecesor a es igual a:

$$\Phi(p).$$

De esta forma las producciones con el mismo predecesor pueden ser aplicadas a varias ocurrencias de la misma letra en un paso de derivación.

Todas las plantas generadas por el mismo sistema L0 determinístico son idénticas, para poderlas variar en el mismo sistema sin producir variantes *artificales* es necesario introducir variaciones "especimen a - especimen" que preserven los aspectos generales de la planta.

Estas variaciones pueden ser alcanzadas por ciertos *valores aleatorios* en el sistema de Lindenmayer. aplicando las producciones estocásticamente afectando topológicamente la estructura de la planta.

Como ejemplo, considérese el siguiente sistema de Lindenmayer estocástico y no determinístico, donde los valores arriba de la flecha indican la probabilidad:

- *alfabeto*:

$$F, [,], +, -$$

- *reglas de producción*:

$$\begin{array}{lll} F & \rightarrow^{.33} & F[+F]F[-F]F \\ F & \rightarrow^{.33} & F[+F]F \\ F & \rightarrow^{.34} & F[-F]F \end{array}$$

- *axioma*:

$$a$$

Algunas variaciones a las plantas representadas por este sistema pueden verse gráficamente de la siguiente forma en la figura 4.11:

Figura 4.11: Sistema de Lindenmayer Estocástico.

4.9 Sistemas de Lindenmayer Paramétricos (LPr)

Los sistemas L Paramétricos operan sobre *palabras paramétricas*, que son cadenas de módulos formados por símbolos pertenecientes a un alfabeto **V**, los cuales tienen parámetros asociados que pertenecen al conjunto de los números reales **R**.

Un módulo con símbolos

$$A \in V$$

y parámetros

$$a_1, a_2, a_3 \ldots, a_n \in R$$

es denotado por

$$A(a_1, a_2, a_3 \ldots, a_n)$$

Todo módulo pertenece al conjunto

$$M = V \times R^*$$

es el conjunto todas las cadenas de módulos no vacías y se denotan como

$$(V \times R) \text{ y} (V \times R)^+$$

respectivamente.

Los parámetros actuales evaluados (números reales) apareciendo en las cadenas, corresponden con los parámetros formales usados en la especificación de las producciones del sistema L. Los parámetros formales pueden existir en expresiones matemáticas y lógicas.

Si Σ es un conjunto de parámetros formales, entonces $C(\Sigma)$ denota a la expresión lógica con parámetros de Σ, y $E(\Sigma)$ representa a una expresión aritmética, con parámetros del mismo conjunto.

Ambos tipos de expresiones consisten de parámetros formales y constantes numéricas combinadas usando los operadores aritméticos

$$+, -, *, /$$

el operador de exponenciación, los operadores relacionales

$$<, >, =$$

y los operadores lógicos

$$!, \&, |$$

y los paréntesis

$$(,)$$

Se siguen las reglas estándar para construir expresiones correctas y para la precedencia de operadores. Las expresiones lógicas evalúan a cero para falso y uno para verdadero.

4.10 Sistemas L Independientes al Contexto Parámetricos (L0Pr)

Uno de los tipos más sencillos de los sistemas L parámetricos son los sistemas L Independientes del contexto.

Estos sistemas están definidos por la cuádrupla

$$G =< V, \Sigma, \omega, P >$$

donde

V	es el alfabeto del sistema,
Σ	es el conjunto de parámetros formales,
$\omega \in (V \times R)^+$	es una palabra paramétrica no vacía llamada *axioma*.
$P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times E(\Sigma))^*$	es un conjunto finito de *producciones*.

Los símbolos ":" y " $\rightarrow$" se usan para separar los tres componentes de una producción: el *predecesor*, la *condición* y el *sucesor*.
Por ejemplo, en la producción:

$$A(t) : t > 5 \rightarrow B(t+1)CD(t^{0.5}, t-2)$$

se tiene que

el *predecesor es*	$A(t)$,
la *condición es*	$t > 5$,
y el *sucesor es*	$B(t+1)CD(t^{0.5}, t-2)$

Una producción concuerda con un módulo en una palabra paramétrica si se cumplen los siguientes puntos:

- El símbolo en el módulo y el símbolo en el predecesor de la producción es el mismo.
- El número de parámetros actuales en el módulo es igual al número de parámetros formales en el predecesor de la producción.
- La condición es evaluada como verdadera, al sustituir los parámetros actuales en los parámetros formales de la producción.
- Una producción puede ser aplicada al módulo, creando una cadena de módulos especificada por el sucesor de la producción. Los valores de los parámetros actuales se sustituyen por los parámetros formales de acuerdo a su posición.

En el caso de que ninguna producción de P concuerde con un módulo, se asume que éste se reemplaza por sí mismo.

Si G es un sistema L y un módulo a produce una palabra paramétrica X como resultado de la aplicación de una producción, se denota como $a \rightarrow X$. Dada una palabra paramétrica

$$\mu = a_1.a_2\ a_3 \ldots a_m$$

se dice que la palabra

$$\nu = X_1.X_2.X_3.. \quad X_m$$

es *directamente derivada de (o generada por)* μ que se denota como $\mu \Rightarrow \nu$ si y sólo si

$$a_i \rightarrow X_i \ \forall\ i = 1,2,\ldots m$$

Una palabra paramétrica ν es generada por G en una *derivación de longitud* n si existe una secuencia de palabras $\mu_0.\mu_1.\ldots.\mu_n$ tal que

$$\mu_0 = \omega, \quad \mu_n = \nu \quad \text{y} \quad mu_0 \Rightarrow mu_1 \Rightarrow mu_2 \Rightarrow . \ . \Rightarrow \mu_n.$$

Por ejemplo, sea el sistema definido por

- *alfabeto:*

$$\Sigma = \{A, B\}$$

- *reglas de producción:*

$$\begin{array}{lllll} p_1: & A(x.y) & : y \leq 3 & \rightarrow A(x*2.x+y), \\ p_2: & A(x,y): & y > 3 & \rightarrow B(x)A(x/y,0), \\ p_3: & B(x): & x < 1 & \rightarrow C. \\ p_4: & B(x): & x \geq 1 & \rightarrow B(x-1) \end{array}$$

- *axioma:*

$$\omega = B(2)A(4,4)$$

Entonces para la primer palabra generada a partir del axioma $\omega = B(2)A(4.4)$, se empieza la reescritura de la siguiente forma:

- para el símbolo B_2 en el módulo, se tiene que cumple como predecesor la condición de la producción p_4 entonces, de acuerdo al valor de $x = 2$ en el módulo, se obtiene el sucesor de $B(2-1) = B(1)$, y para el símbolo $A(x = 4.y = 4)$ en el módulo, se tiene que cumple con la condición de la propiedad p_2, entonces se obtine el sucesor $B(4)A(4/4.0)$. La primer palabra generada es $B(1)B(4)A(1.0)$. A partir de esta palabra se puede obtener la siguiente palabra reescrita tomando los valores actuales de x y de y, empezando a reescribir $B(1)$, se obtine que cumple con la condición de la regla p_4, entonces $B(1)$ evoluciona a $B(1-1) = B(0)$, para $B(4)$, cumple igualmente con la condición de la regla p_4 y evoluciona a $B(4-1) = B(3)$, y para $A(1.0)$ se ve que cumple con la condición de la propiedad p_1, entonces evoluciona a $A(1*2.1+0) = A(2.1)$. Aplicando sucesivamente este proceso de reescritura se obtiene que la cuarta palabra generada es $CB(1)A(8.7)$. Este proceso de reescritura, no debe de olvidarse se realiza en forma simultánea para todos los módulos.

La secuencia inicial de generación de este sistema 0L Paramétrico está representada en la figura 4.12:

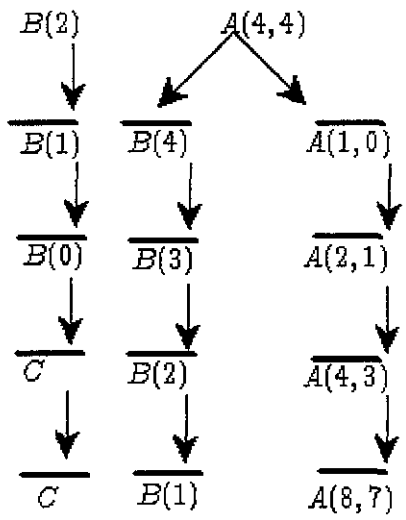


Figura 4.12: Secuencia de generación de un sistema L Paramétrico.

4.11 Sistemas de Lindenmayer Sensibles al Contexto (LI)

En todos los sistemas de Lindenmayer discutidos hasta ahora las células individuales se desarrollan sin ninguna interacción con sus vecinos. En términos de teoría de lenguajes, la reescritura ha sido independiente del contexto. Sin embargo, en muchas ocaciones se encuentran organismos en los cuales cada una de sus células está influenciada por el estado de algunas de sus vecinas para el proceso de reescritura. Es decir, cada célula depende de células vecinas para poder pasar a la siguiente etapa. Estos sistemas son conocidos como *sistemas de Lindenmayer con intercciones* o *sistemas de Lindenmayer Sensibles al Contexto.*

En un sistema $L(m,n)$ con:

$$m,n \geq 0$$

las reglas son de la forma:

$$(\alpha,a,\beta) \rightarrow w, \quad |\alpha| = m, |\beta| = n$$

Esto significa, que entre las letras α y β, la letra a puede ser reescrita como la palabra w.

En este tipo de sistemas también se aplica el *paralelismo*: todas las letras deben de ser reescritas simutáneamente.

El nombre de *sistemas de LindenmayerI* es colectivo para todos los sistemas L(m,n).

Los sistemas $L(1,0)$ y $L(0,1)$ son llamados *sistemas 1L*: la interacción de la célula es de un lado. la reescritura de una letra depende siempre de su vecino izquierdo solamente (o de su vecino derecho).

El uso de las letras D.P E. tienen el mismo significado anterior. En particular determinismo significa en este caso, que para cada configuración que consiste de una letra y una vecindad (m.n), hay exactamente una regla

Como ejemplo sea el siguiente sistema L(1.0)D.

- *alfabeto*:
$$\{a, b, c, d\}$$
- *reglas de producción*:
las reglas están definidas en la siguiente tabla, donde cada renglón indica la vecindad izquierda y cada columna, la letra a ser reescrita:

	a	b	c	d
#	c	b	a	d
a	a	b	a	d
b	a	b	a	d
c	b	c	a	ad
d	a	b	a	d

- *axioma*:
$$ab$$

Debido al determinismo el leguaje es producido en una *secuencia* con comienzo:

$$ad, cd, aad, cad, abd, cbd, acd, caad, abad, cbad, acad, cabd, abbd, cbbd, acbd, cacd, abaad, \ldots$$

Este crecimiento en la longitud de palabra no es posible para las secuencias generadas por sistemas de Lindenmayer sin interacción o sistemas L0D.

Los intervalos en los cuales la función de crecimiento permanece constante crece más allá de todos los límites. El crecimiento es logarítmico.

Es obvio que la capacidad generativa aumenta si la interacción es más extensiva, los sistemas:

$$L(m+1, n)$$

son más generativos que los sistemas L(m n).

Los sistema $L(0, 1)E$ (también como los sistemas $L(1\ 0)E$) generan todos los lenguajes recursivamente enumerables.

4.12 Problemas de Decisión.

Sea un sistema 0L $G = \{\Sigma, h\ \omega\}$ que genera sistemáticamente elementos de $L(G)$ de la siguiente forma.

$$L(G) = \cup_{n=0}^{\infty} L_n(G).$$

- Sea $L_0(G)$ el lenguaje que tiene exactamente un elemento, que es ω.
- Para cualquier n, si ya se han generado todos los elementos del lenguaje $L(G)$, éstos pueden ser listados en una secuencia.

Sin embargo, el lenguaje $L(G)$ puede ser infinito, en este caso el proceso para generar cadenas del sistema G nunca termina. Además si se da una cadena arbitraria $x \in \Sigma^*$, para saber si pertenence al lenguaje $L(G)$ es imposible hacer el proceso de listar los elementos de $L(G)$, y por lo tanto no permite saber si $x \in L(G)$.

Es muy importante tener un procedimiento (o algoritmo) efectivo el cual, dado un sistema 0L G y una palabra x, decida en un número finito de pasos si x está o no en $L(G)$. Este problema es conocido como el *problema de pertenencia*.

En general hay dos problemas de decisión relacionados con los sistemas 0L:

- *Problema de Pertenencia*:
  para cualquier sistema 0L arbitrario y cualquier palabra arbitraria x, decidir si x está o no en $L(G)$.
- *Problema de Equivalencia*:
  para cualquiera dos sistemas 0L G_1 y G_2 decidir si $L(G_1) = L(G_2)$ o no.

Dos sistemas G_1 y G_2 tal que $L(G_1) = L(G_2)$ son *equivalentes* y esta equivalencia se denota por:

$$G_1 \sim G_2$$

Una forma de resolver los problemas de decisión consiste en describir un algoritmo. Si este algoritmo existe el problema de decisión se dice *soluble*, de lo contrario es *no soluble*. Entonces, se tiene que:

- El problema de pertenencia es soluble.
- El problema de equivalencia es no soluble.

4.13 Derivaciones y Gráficas de Derivación en Sistemas 0L.

Intuitivamente, se entiende por derivación y de x, una secuencia de palabras que empieza con la palabra x y termina con la palabra y, aplicando alguna de las producciones del conjunto de producciones en cada paso.

Sea $G = \{\Sigma, h, \omega\}$ un sistema 0L. Una *derivación* D en G es una terna

$$(\theta, \nu, p)$$

donde:

- θ es un conjunto finito de pares ordenados de enteros no negativos.
- ν es una función de θ sobre Σ tal que:

  $$\nu(i, j) \text{ es el valor de } D \text{ en la ocurrencia } (i, j).$$

- p es una función de θ sobre $\cup_{a \in \Sigma}\{a \to \alpha \mid \alpha \in h(a)\}$ tal que:

  $$p(i, j) \text{ es la producción de } D \text{ en la ocurrencia } (i, j).$$

La *ocurrencia* (i, j) satisface las siguientes condiciones:

- existe una secuencia de palabras $(x_0, x_1, \ldots, x_r) \in \Sigma^*$, llamada la *traza de* D tal que $r \geq 1$ y:

1. $\theta = \{(i, j) \mid 0 \leq i < r \quad \text{y} \quad 1 \leq j \leq |\, x_i \,|\}$:
2. $\nu(r, j)$ es el j-ésimo símbolo en x_r.
3. $\forall 0 \leq i < r. \quad x_{i+1} = \alpha_1 \alpha_2 \ldots \alpha_{r_i}. \quad$ donde $\quad p(i, j) = (\nu(i, j), \alpha_j) \quad$ para $\quad a \leq j \leq |\, x_i \,|$.

En este caso D es una *derivación* de x_r a partir de x_0 y r es la longitud de D. La cadena x_r es llamada el *resultado* de una derivación D. En particular si $x_0 = \omega$, entonces, D es una *derivación* de x_r en G.

Por ejemplo, sean los sistemas 0L:

$$1.G = \{\{a\}, \{a \to a,\ a \to a^2\}, a^2\}$$

y sea

$$D = (\theta, \nu, p)$$

donde

- $\theta = \{(0, j) \mid 1 \leq j \leq 2\} \cup \{(1, j) \mid 1 \leq j \leq 3\}$.
- $\nu = (i, j) = a$ para $(i, j) \in \theta$.
- $p(0, 1) = p(1, 2) = a \to a$ y $p(0, 2) = p(1, 1) = p(1, 3) = a \to a^2$.

entonces D es una derivación de a^5 en G de longitud 2.

$$2.G = \{\{a, b\}, \{a \to (ab)^2, b \to \lambda\}, ab\}$$

entonces:

$$D = (\theta, \nu, p)$$

donde

- $\theta = \{(0\ 1), (0, 2)\} \cup \{(1, j) \mid 1 \leq j \leq 4\} \cup \{(2\ j) \mid 1 \leq j \leq 8\}$.
- $\nu(r, j) = \begin{cases} a & \text{si } j \text{ es impar} \\ b & \text{si } j \text{ es par} \end{cases}$
- $p(i, j) = \begin{cases} a \to & (ab^2) & \text{si } j \text{ es impar} \\ b \to & \lambda & \text{si } j \text{ es par} \end{cases}$

es una derivación de $(ab)^4$ en G.

Dado un sistema 0L y una derivación D, una forma sencilla de representar D es a través de una gráfica llamada *Gráfica de Derivación* de D.

Por ejemplo sea el sistema 0L:

$$G = \{\{a\}, \{a \to a^2, a \to a^3\}\}$$

y sean D_1 y D_2:

$$D_1 = (\theta, \nu, p_1) \quad \text{y} \quad D_2 = (\theta, \nu, p_2)$$

donde:

- $\theta = \{(0,1), (0,2)\} \cup \{(1,j) \mid 1 \leq j \leq 5\}$.
- $\nu(i,j) = a \forall (i,j) \in \theta$.
- $p_1(0,1) = (a, aa)$, $p_1(0,2) = (a, aaa)$.
- $p_2(0,1) = (a, aaa)$, $p_2(0,2) = (a, aa)$.

entonces D_1 y D_2 son dos derivaciones diferentes de a^5 a partir de a^2, ambas son de longitud 1.

Estas dos derivaciones son representadas por las gráficas de derivación de la figura 4.13:

Figura 4.13: Gráficas de derivación 1

Las derivaciones y los árboles de derivación son correspondencias *uno a uno* esto es, dada una derivación D se puede construir únicamente una gráfica de derivación T que representa a D y de igual forma, para una gráfica de derivación T sólo se puede construir la derivación D que está representada por T.

A través de las gráficas de derivación se puede observar que cada ocurrencia (i,j) en una derivación D determina una única subderivación de una subcadena x_f de $\nu(i,j)$. Sea:

$$D(i,j) \text{ denota la subderivación determinada por } (i,j),$$

y sea:

$$y \text{ tal que } D(i,j) \text{ de } y \text{ desde } \nu(i,j).$$

Por ejemplo, $y = \rho(D(i,j))$. Pueden distinguirse dos posibilidades:

1. $y \neq \lambda$ en este caso la longitud de $D(i,j)$ es exactamente $f - i$, y (i,j) es una ocurrencia *productiva* en D.
2. $y = \lambda$ en este caso la longitud de $D(i,j)$ es menor o igual a $f - i$, y (i,j) es una ocurrencia *improductiva* de D.

Para la última gráfica de derivación (1, 1) y (1, 3) son ocurrencias productivas. la gráfica de derivación de $D(1\ 1) = D(1, 3)$ está en la figura 4.14:

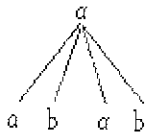


Figura 4.14: Gráfica de derivación 2.

Por otro lado, (0, 2) es una ocurrencia productiva y la gráfica de derivación de $D(0\ 2)$ es la mostrada en la figura 4.15:

Figura 4.15: Gráfica de derivación 3.

Sea G un sistema L0 y sea $D = (\theta, \nu, p)$ y E derivaciones en G de longitudes f y g respectivamente. Además el primer elemento de la traza de E es un solo símbolo a.
Para cualquier ocurrencia $(i\ j)$ en D, se puede reemplazar $D(i, j)$ por E y obtener una nueva derivación en S en la que:

1. $\nu(i, j) = a$.
2. $f - i = g$ o $f - i \geq g$ y E es una derivación de λ desde a.

Por ejemplo, sea G el sistema anterior:

$$G = \{\{a\}, \{a \to a^2, a \to a^3\}\}$$

y sean D_1 y D_2:

$$D_1 = (\theta, \nu, p_1) \quad \text{y} \quad D_2 = (\theta, \nu, p_2)$$

donde:

- $\theta = \{(0, 1), (0, 2)\} \cup \{(1, j) \mid 1 \leq j \leq 5\}$.
- $\nu(i, j) = a \forall (i, j) \in \theta$.
- $p_1(0, 1) = (a, aa)$, $p_1(0, 2) = (a, aaa)$.
- $p_2(0, 1) = (a, aaa)$, $p_2(0, 2) = (a, aa)$

y sea D la gráfica de derivación de la figura 4.16:

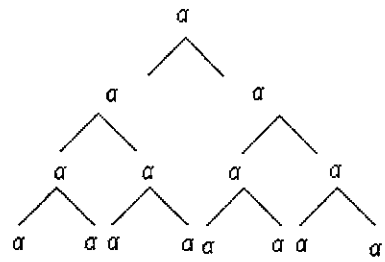


Figura 4.16: Gráfica de derivación 4.

y sea E la gráfica de derivación de la figura 4.17:

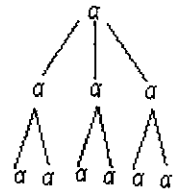


Figura 4.17: Gráfica de derivación.

entonces se puede sustituir E por $D(1,2)$ y obtener una gráfica de derivación válida cuya gráfica es la mostrada en la figura 4.18:

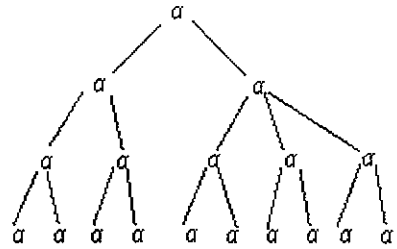


Figura 4.18: Gráfica de derivación 5.

4.14 Filtros

Un rasgo muy característico de los sistemas L0 es que tienen que aceptar todo lo que producen. Se tiene el axioma, las reglas de producción, y se desea modelar un fenómeno. Sin embargo se puede dar el caso en que se quiera *excluir* algunos resultados generados por las reglas de producción, porque producirían una degeneración al fenómeno. Esto no es posible y se tiene que aceptar toda la configuración resultante. No hay forma de eliminar las palabras no deseadas, es decir no hay mecanismos de filtro.

La teoría de lenguajes formales comúnmente usa varios *mecanismos de filtro*. No todas las palabras obtenidas en la derivación son consideradas en el lenguaje. el lenguaje terminal es algo *excluyente* con respecto a todo el conjunto de las palabras derivadas.

El ejemplo más típico entre los mecanismos de filtro, completamente esencial en las gramáticas de Chomsky es el uso de *letras no terminales* . Una ocurrencia de un *no terminal* en una palabra significa que la palabra no es aceptable (o al menos no todavía). El lenguaje generado contiene sólo palabras con no terminales. Entonces el lenguaje obtenido directamente es el que consiste en no terminales. Esto da la posibilidad de *rechazar* (al menos algunas) palabras no deseadas.

Este mismo mecanismo, como muchos otros filtros, pueden ser usados en los sistemas de Lindenmayer. Sin embargo se debe de tener cuidado, ya que el objetivo original fue modelar el desarrollo de las especies en forma real o artificial. Cada palabra generada se supone que representa una etapa en el desarrollo Sería arbitrario excluir algunas palabras y decir que no representan etapas del desarrollo.

Algunos experimentos más cuidadosos u observaciones pueden cambiar la interpretación de células individuales para representar alguna fase en el desarrollo, después del cual las células son renombradas. Esto implica la aplicación de morfismos letra a letra al lenguaje. Por este resultado el uso de los no terminales es muy bien visto.

Los mecanismos de filtro proporcionan a las familias de los lenguajes L una característica muy deseable en ambos sentidos matemáticamente y en teoría de lenguajes, esta es la propiedad de *clausura* bajo ciertas operaciones. Sin filtros, las familias *puras* como son los lenguajes L0 y los L0D tienen propiedades de clausura muy débiles, la mayoría de las operaciones de teoría de lenguajes producen lenguajes fuera de esa familia.

Una variación más de los sistemas de Lindenmayer en tener un *conjunto finito de axiomas*, en lugar de un solo axioma, como en las definiciones anteriores. Esta variación se nombra en los sistemas de Lindenmayer añadiendo la letra F.

4.15 Adultos

En alguna etapa los organismos modelados por sistemas de Lindenmayer, se espera que lleguen a ser *adultos*. Se acostumbra definir a los adultos en la teoría de los sistemas de Lindenmayer de la siguiente manera

Una palabra pertenece al *lenguaje de adultos* de un sistema en el caso exacto que no deriva palabras de si mismo. Por ejemplo.

$$a \to ab. \quad b \to c, \quad c \to \lambda \quad d \to dc$$

son las únicas reglas para a, b, c, d en un sistema L0 Entonces todas las palabras de la forma

$$(abc)^i(dc)^j$$

pertenecen al lenguaje adulto. Los lenguajes adultos de sistemas son llamados *lenguajes L0A*. Por supuesto, A puede ser usada con otros tipos de sistemas de Lindenmayer.

Hay un hecho interesante entre la teoría de lenguajes que L0A es igual al la familia de los lenguajes independientes del contexto. Similarmente L(1 0)A es igual a la familia de los lenguajes recursivamente enumerables.

4.16 Fragmentación

Los sistemas L *con fragmentación, sistemas LJ*, son úitles para modelar vida artifical. El mecanismo de fragmentación proporciona un nuevo formalismo para comunicación en bloques, dividiendo el desarrollo de un filamento y a la célula muerta.

La idea básica del mecanismo de fragmentación es la siguiente. los lados derechos de las reglas pueden contener ocurrencias de un símbolo especial q. Este símbolo induce a un *corte* en la palabra examinada. y la derivación puede continuar de cualquiera de las partes obtenidas. Por ejemplo si aplicamos las reglas

$$a \to aqa, \quad b \to ba, \quad c \to qb$$

a la palabra abc. obtenemos las palabras a, aba, y b. La derivación puede continuar de cualquiera de ellas. El sistema L0J con el axioma aba y las reglas

$$a \to a, \quad b \to abaqaba$$

genera el lenguaje

$$\{aba^n \mid n \geq 1\} \cup \{a^n ba \mid n \geq 1\}.$$

Un punto importante en la teoría de lenguajes con los sistemas L0E, además de muchas otras propiedades de clausura es que son cerradas bajo la fragamentación:

$$L0JE = L0EJ = L0E.$$

Los sistemas L0J han sido utilizados recientemente para obtener representaciones compactas de ciertos árboles regulares.

En conclusión, se puede obtener el *glosario* mostrado en la tabla 4.2:

Símbolo	Definición
A	adulto, palabra adulta, lenguaje adulto
B	bifurcado, ramificación
D	determinístico, opción única, solo una opción de cada tabla
E	extendido, intersección con un vocabulario terminal
Fl	florecente, que florece o da fruto
F	conjunto finito de axiomas, más de un sólo axioma
I	interacción, células vecinas afectan el desarrollo de la célula
J	fragmentación, mecanismo para crear cortes
L	Lindenmayer, aparece en el nombre de todos los sistemas
N	sustitución sin eliminación
O	realmente representa el número 0, pero frecuentemente se lee como una letra información nula de la interacción por ninguno de los lados, reescritura independiente del contexto
P	propagado, no hay célula muerta, la palabra vacía nunca aparece del lado derecho de la producción
S	estocástico, sujeto a probabilidad
T	tablas, diversos conjuntos de reglas, diversificación de instrucciones de desarrollo
U	unario, alfabeto que consiste en una sola letra

Tabla 4.2: Glosario.

Capítulo 5

Conclusiones

En este trabajo concurrieron diferentes áreas, la teoría de lenguajes formales, los sistemas de reescritura y las propiedades y fundamentos de los sistemas de Lindenmayer.

Para la generación de vida artificial, se pueden utilizar métodos basados en dispositivos generativos, estos dispositivos son estudiados dentro de la teoría de lenguajes formales, los principales dispositivos generativos dentro de esta teoría son los autómatas celulares, las máquinas de Turing y las gramáticas. A partir de las gramáticas formales se puede cambiar la forma en que se desarrolla la reescritura, (ésta es siempre secuencial en la teoría de lenguajes formales), por una reescritura en paralelo ya que ésta permite la simulación de procesos de cambio en organismos vivos.

El diagrama de la figura 5.1 ilustra de mejor forma esta relación.

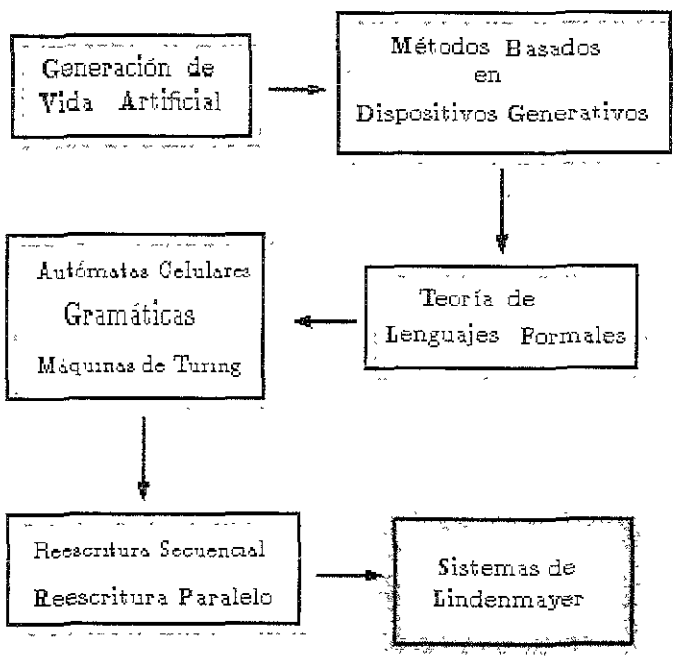


Figura 5.1: Contexto teórico de los sistemas L.

Los sistemas de Lindenmayer son un caso especial de gramáticas formales con reescritura en paralelo. Es posible establecer una caracterización de estos sistemas a partir de la teoría de lenguajes formales, de acuerdo a las propiedades intrínsecas de cada sistema, como son la interacción o no interacción entre las células, el determinismo, el no determinismo, la propagación, el florecimiento, la ramificación, la extención del alfabeto, multiplicidad de tablas de reglas de reescritura, parametrización etc.

La figura 5.2 muestra una caracterización de los principales sistemas de Lindenmayer estudiados en este trabajo de tesis, para exhibir su relación.

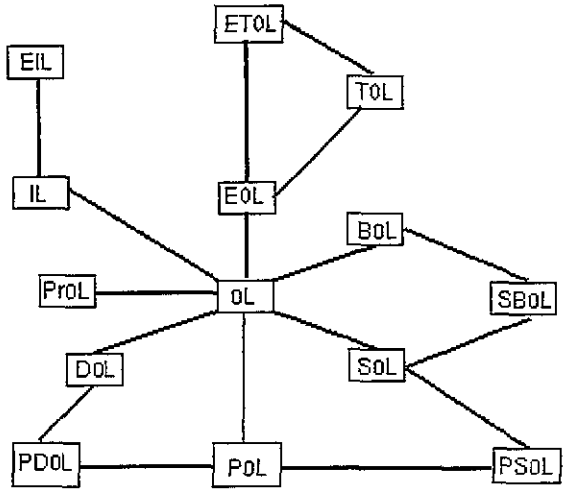


Figura 5.2: Caracterización de los sistemas L.

En la parte superior izquierda, están representados los sistemas L sensibles al contexto (IL), un caso particular de estos sistemas es cuando el predecesor no presenta vecinos en su lado izquierdo ni en el derecho, en este caso se habla de sistemas L independientes del contexto (0L), estos a su vez, pueden representar diferentes caractrísticas y crear tipos de sistemas independientes al contexto más específicos. Por ejemplo, en la figura 5.2, ubicándose sobre los sistemas L independientes al contexto y siguiendo el sentido de las manecillas del reloj, se pueden identificar por ejemplo, los sistemas:

- Sistemas *E0L* Sistemas independientes del contexto con dos alfabetos disjuntos, símbolos terminales y no terminales.
- Sistemas *B0L* Sistemas independientes del contexto con símbolos [,] en su alfabeto para denotar ramificaciones.
- Sistemas *S0L* Sistemas independientes del contexto con un conjunto de reglas de producción estocásticas o con probabilidad asignada. Ahora bien, a partir de este tipo de sistemas se pueden obtener otros más específicos como pueden ser:
  - Sistemas *SB0L* que son sistemas independientes del contexto, bifurcados y además estocásticos.
  - Sistemas *PS0L* que son sistemas independientes del contexto, estocásticos y propagados.

En general se pueden ir uniendo diferentes propiedades, para obtener sistemas de Lindenmayer, mucho más específicos.

Las principales aportaciones de este trabajo son las siguientes.

- Presenta a los sistemas de Lindenmayer dentro del contexto de los lenguajes formales.
- Muestra que su modelación es en efecto, muy simple.
- Proporciona claros ejemplos del funcionamiento del mecanismo de reescritura en paralelo
- Sitúa a los sistemas de Lindenmayer en el contexto de los sistemas de reescritura más poderosos utilizados en el campo de la computación científica.
- Proporciona una clara caracterización de estos sistemas, sin ambigüedades para un modelado sencillo.
- La caracterización, brinda facilidades para el desarrolo de software.
- Proporciona las bases teóricas fundamentales para el estudio de sistemas de Lindenmayer contínuos.
- Establece claramente los conceptos de la reescritura en paralelo para facilitar una implementación de software multihilada.

5.1 Trabajos Posteriores

Este trabajo de tesis tiene como objetivo proponer a los sistemas L como generadores de vida artificial, explicar su funcionamiento, sus propiedades básicas y establecer una caracterización entre ellos. Los trabajos posteriores a este son el desarrollo del software que permita interpretar geométricamente cada una de las cadenas y genere las imagenes del proceso de desarrollo para la simulación.

Lista de Tablas

Lista de Figuras

Bibliografía

[1] John E. Hopcroft. *Introduction to Automata Theory Languages and Computation* Addison - Wesley. 1967

[2] Arto Salomaa. *Formal Languges.* Academic Prees New York1973.

[3] Peter Linz. *An Introduction to Formal Languges and Automata.* University of California at Davis 1997

[4] Gleen Brookshear *Theory of Computation-Formal Languages. Automata and Complexity.* Cummings Publishing Company. 1989.

[5] P.Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer Verlag New York.1990

[6] G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems.* Academic Prees New York.1980.

[7] M F. Barnsley. *Fractals Everywhere.* Academic Prees San Diego.1988

[8] M. Gardner. *Mathematical games: An array of Problems that Can Be Solved with Elementary mathematical Techniques.* Scientific American. 216.1967.

[9] G.T Herman and G. Rozenberg. *Developmental Systems and Languages.* North-Holland. Amsterdam 1975.

[10] H. Peitgen and D Saupe. *The Science of Fractal Images.* Springer Verlag. New York1988.

[11] P. Prusinkiewicz and James Hanan. *Lindenmayer Systems. Fractals and Plants* volume 79 of Lecture Notes in Biomathematics. Springer Verlag. 1989.

[12] G. T. Herman and G. Rozenberg. *Developmental Systems and Languages.* North Holland New York1975

[13] G Rozenberg. *Propagating L-Systems with Interactions.* Departament of Computer Science State University. New York1988.