

# TEORÍA DE LA COMPUTACIÓN

Wilber Ramos Lovón



# Índice general

<b>1. Introducción</b>	<b>5</b>
1.1. El papel de los algoritmos . . . . .	5
1.2. Una breve historia de la computación . . . . .	8
1.2.1. Primera Generación . . . . .	14
1.2.2. Segunda Generación . . . . .	17
1.2.3. Tercera generación . . . . .	20
1.2.4. Cuarta Generación . . . . .	23
1.2.5. Quinta Generación . . . . .	25
<b>2. Lenguajes y Gramáticas</b>	<b>27</b>
2.1. Gödel y Turing . . . . .	27
2.2. Autómatas . . . . .	28
2.3. Lenguajes y Gramáticas . . . . .	29
2.4. Máquinas Abstractas y Lenguajes Formales . . . . .	29
2.5. Alfabetos, Símbolos y palabras . . . . .	30
2.6. Operaciones con Palabras . . . . .	32
2.6.1. Operación Cerrada: . . . . .	32
2.6.2. Propiedad Asociativa: . . . . .	32
2.6.3. Existencia de Elemento Neutro: . . . . .	33
2.6.4. Potencia de una Palabra: . . . . .	33
2.6.5. Reflexión de una Palabra: . . . . .	33
2.7. Lenguajes . . . . .	33
2.7.1. Concatenación de Lenguajes . . . . .	34
2.7.2. Clausura Positiva de un Lenguaje . . . . .	34
2.7.3. Iteración, Cierre o Clausura de un Lenguaje . . . . .	35
2.7.4. Reflexión de Lenguajes . . . . .	35
2.8. Conceptos Básicos sobre Gramáticas . . . . .	35
2.8.1. Gramática de Estructura de Frases . . . . .	36
2.8.2. Tipos de Gramáticas . . . . .	39
2.9. Expresiones Regulares . . . . .	41
2.10. Árboles de Derivación . . . . .	43

2.11. Lista de ejercicios . . . . .	43
<b>3. Autómatas Finitos</b>	<b>45</b>
3.1. Autómatas Finitos Deterministas ( <i>AFD</i> ) . . . . .	46
3.2. Autómatas Finitos no Deterministas ( <i>AFND</i> ) . . . . .	49
3.3. Otras Formas del Lema del Bombeo . . . . .	53
3.4. Equivalencia y Minimización de Autómatas Finitos . . . . .	54
3.5. Lista de Ejercicios . . . . .	55
<b>4. Autómatas de Pila y L. I. del Contexto</b>	<b>59</b>
4.1. Forma normal Chomsky . . . . .	73
4.2. Lista de Ejercicios . . . . .	76
<b>5. Máquinas de turing</b>	<b>81</b>
5.1. Conceptos . . . . .	81
5.2. Construcción Modular de Máquinas de Turing . . . . .	84
5.2.1. Bloques de construcción básicos . . . . .	84
5.2.2. Bloques más complejos . . . . .	85
5.3. Evaluación de Cadenas . . . . .	86
5.3.1. Máquinas de Turing de varias cintas . . . . .	87
5.4. Máquinas de Turing no deterministas . . . . .	90
5.5. Lenguajes aceptados por Máquinas de Turing . . . . .	91
5.5.1. Máquinas de Turing universales . . . . .	93
5.5.2. Lenguajes Aceptables y Decidibles . . . . .	94
5.5.3. El Problema de la Parada . . . . .	95
5.6. Lista de Ejercicios . . . . .	95

# Capítulo 1

## Introducción

*La Ciencia de la Computación o Infórmatica es la disciplina que trata de establecer una base científica para temas tales como el diseño asistido por computadora, la programación de computadores, el procesamiento de la información, las soluciones algorítmicas de problemas y el propio proceso algorítmico.*

### 1.1. El papel de los algoritmos

Para que una máquina como una computadora pueda llevar a cabo una tarea, es preciso diseñar y representar un algoritmo de realización de dicha tarea y en una forma que sea compatible con la máquina. A la representación de un algoritmo se le denomina programa. Por comodidad de los seres humanos, los programas computacionales suelen imprimirse en papel o visualizarse en las pantallas de las computadoras. Sin embargo para comodidad de las máquinas, los programas se codifican de una manera compatible con la tecnología a partir de la cual esté construida la máquina. El proceso de desarrollo de un programa en una máquina de denomina programación. Los programas y los algoritmos, representan, lo que colectivamente se denomina

software, y lo que tenga que ver con la propia máquina que se conoce con el nombre de hardware.

El estudio de los algoritmos comenzó siendo un tema del campo de las matemáticas. De hecho, la búsqueda de algoritmos fue una actividad de gran importancia para los matemáticos mucho antes del desarrollo de las computadoras actuales. El objetivo era determinar un único conjunto de instrucciones que describiera como resolver todos los problemas de un tipo concreto. Uno de los ejemplos mejor conocido de estas investigaciones pioneras es el algoritmo de división para el cálculo del cociente de dos números de varios dígitos. Otro ejemplo es el algoritmo de Euclides<sup>1</sup>



Figura 1.1: Pintura idealizada de Euclides.

---

<sup>1</sup>fué un matemático y geómetra griego (ca. 325-ca. 265 a.c), a quien se le conoce como "El padre de la geometría". Su vida es poco conocida, salvo que vivió en Alejandría (actualmente Egipto) durante el reinado de Ptolomeo I. Ciertos autores árabes afirma que euclides era hijo de Naucrates y se barajan tres hipótesis:

- Euclides fué un personaje matemático histórico que escribió *Los elementos y obras atribuidas a él*.
- Euclides fué el líder de un equipo de matemáticos que trabajaba en Alejandría.
- Las obras completas de Euclides fueron escritas por un equipo de matemáticos de Alejandría quienes tomaron el nombre de Euclides

Una vez que se ha encontrado el algoritmo para llevar a cabo una determinada tarea, la realización de esta ya no requiere comprender los principios en los que el algoritmo está basado. En cierto sentido, la inteligencia requerida para resolver ese problema está codificada dentro del algoritmo.

Es esta capacidad de capturar y transmitir inteligencia por medio de algoritmos, lo que nos permite construir máquinas que lleven a cabo tareas de utilidad. En consecuencia, el nivel de inteligencia mostrado por las máquinas está limitado por la inteligencia que podamos transmitir mediante algoritmos. Solo podemos construir una máquina para llevar a cabo una tarea si existe un algoritmo que permita realizar esa tarea. A su vez, si no existe ningún algoritmo para resolver un problema, entonces la solución de ese problema cae fuera de la capacidad de las máquinas disponibles.

La identificación de las limitaciones de las capacidades algorítmicas terminó convirtiéndose en uno de los temas de las matemáticas en la década de 1930 con la publicación del teorema de incompletitud de Kurt Gödel<sup>2</sup>.



Figura 1.2: Kurt Gödel.

---

<sup>2</sup>Kurt Gödel (28 de abril de 1906 Brünn, Imperio austrohúngaro, actual República Checa – 14 de enero de 1978, Princeton, Estados Unidos) fue un lógico, matemático y filósofo austriaco-estadounidense. Reconocido como uno de los más importantes lógicos de todos los tiempos, el trabajo de Gödel ha tenido un impacto inmenso en el pensamiento científico y filosófico del siglo XX.

Este teorema afirma, en esencia, que en cualquier teoría matemática que abarque nuestro sistema aritmético tradicional, hay enunciados cuya verdad o falsedad no puede establecerse por medios algorítmicos.

El enunciado de este hecho removió los cimientos de las matemáticas y el estudio de las capacidades algorítmicas que se inicio a partir de ahí fué el comienzo del campo que hoy día conocemos como Ciencia de la Computación. De hecho, es el estudio de los algoritmos lo que forma la base fundamental de estas ciencias.

## 1.2. Una breve historia de la computación

Las computadoras actuales tienen una genealogía muy extensa. Uno de los primeros dispositivos mecánicos para contar fue el ábaco, cuya historia se remonta a las antiguas civilizaciones griega y romana.

Este dispositivo es muy sencillo, consta de cuentas ensartadas en varillas que a su vez están montadas en un marco rectangular. Al desplazar las cuentas sobre varillas, sus posiciones representan valores almacenados, y es mediante dichas posiciones que este representa y almacena datos. A este dispositivo no se le puede llamar computadora por carecer del elemento fundamental llamado programa.



Figura 1.3: ábaco romano.

Otro de los inventos mecánicos fue la Pascalina inventada por Blaise Pascal (1623 - 1662) de Francia y la de Gottfried Wilhelm von Leibniz (1646 - 1716) de Alemania. Con estas máquinas, los datos se representaban mediante las posiciones de los engranajes, y los datos se introducían manualmente estableciendo dichas posiciones finales de las ruedas, de manera similar a como leemos los números en el cuentakilómetros de un automóvil.



Figura 1.4: Pascalina en el Museo de Artes y Oficios de París.

La primera computadora fue la máquina analítica creada por Charles Babbage, profesor matemático de la Universidad de Cambridge en el siglo XIX. La idea que tuvo Charles Babbage sobre un computador nació debido a que la elaboración de las tablas matemáticas era un proceso tedioso y propenso a errores. En 1823 el gobierno Británico lo apoyó para crear el proyecto de una máquina de diferencias, un dispositivo mecánico para efectuar sumas repetidas.



Figura 1.5: La máquina analítica de Babbage, como se puede apreciar en el Science Museum de Londres.

Mientras tanto Charles Jacquard (francés), fabricante de tejidos, había creado un telar que podía reproducir automáticamente patrones de tejidos leyendo la información codificada en patrones de agujeros perforados en tarjetas de papel rígido. Al enterarse de este método Babbage abandonó la máquina de diferencias y se dedicó al proyecto de la máquina analítica que se pudiera programar con tarjetas perforadas para efectuar cualquier cálculo con una precisión de 20 dígitos. La tecnología de la época no bastaba para hacer realidad sus ideas.



Figura 1.6: El telar de Charles Jacquard.

El mundo no estaba listo, y no lo estaría por cien años más. En 1944 se construyó en la Universidad de Harvard, la Mark I, diseñada por un equipo encabezado por Howard H. Aiken. Esta máquina no está considerada como computadora electrónica debido a que no era de propósito general y su funcionamiento estaba basado en dispositivos electromecánicos llamados relevadores.

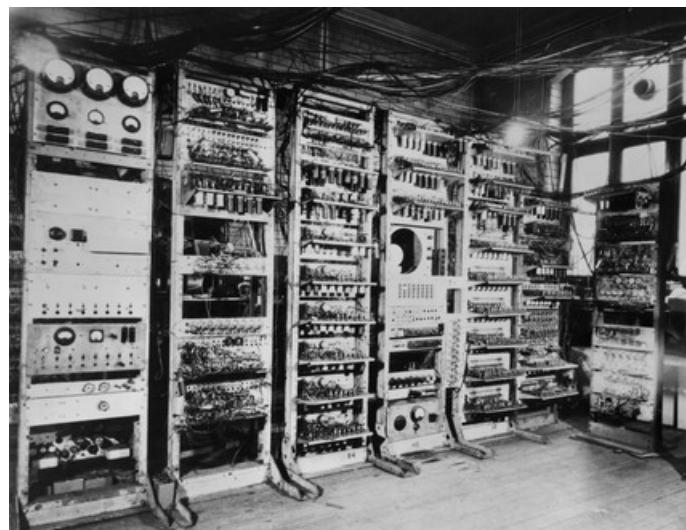


Figura 1.7: Mark I, fotografía de 1948.

En 1947 se construyó en la Universidad de Pennsylvania la ENIAC (Electronic Numerical Integrator And Calculator) que fue la primera computadora electrónica, el equipo de diseño lo encabezaron los ingenieros John Mauchly y John Eckert. Esta máquina ocupaba todo un sótano de la Universidad, tenía más de 18 000 tubos de vacío, consumía 200 KW de energía eléctrica y requería todo un sistema de aire acondicionado, pero tenía la capacidad de realizar cinco mil operaciones aritméticas en un segundo.

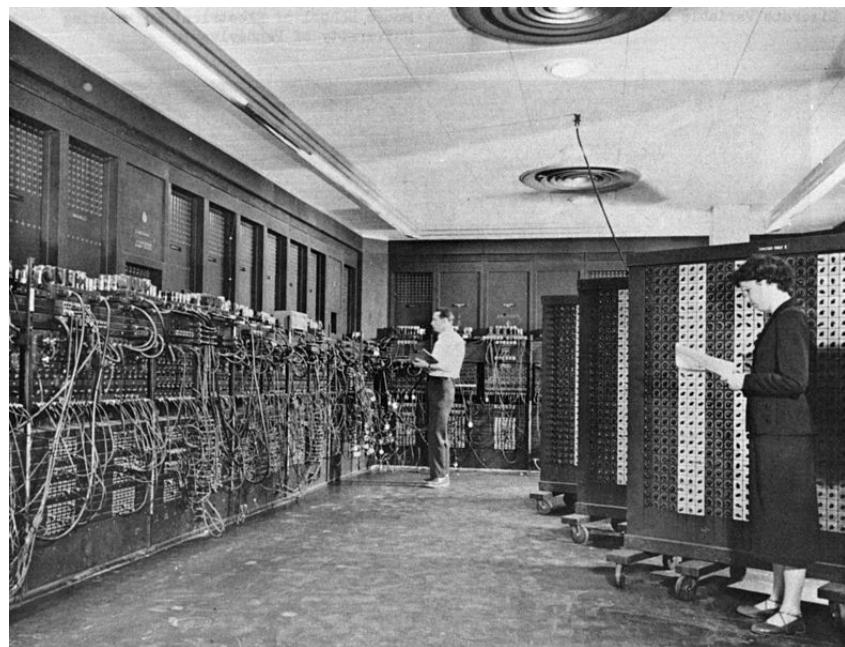


Figura 1.8: Glen Beck (background) and Betty Snyder (foreground) program ENIAC in BRL building 328. (U.S. Army photo).

El proyecto, auspiciado por el departamento de Defensa de los Estados Unidos, culminó dos años después, cuando se integró a ese equipo el ingeniero y matemático húngaro John von Neumann (1903 - 1957). Las ideas de von Neumann resultaron tan fundamentales para su desarrollo posterior, que es considerado el padre de las computadoras.

La EDVAC (Electronic Discrete Variable Automatic Computer) fue diseñada por este nuevo equipo. Tenía aproximadamente cuatro mil bulbos y usaba un tipo de memoria basado en tubos llenos de mercurio por donde circulaban señales eléctricas sujetas a retardos.

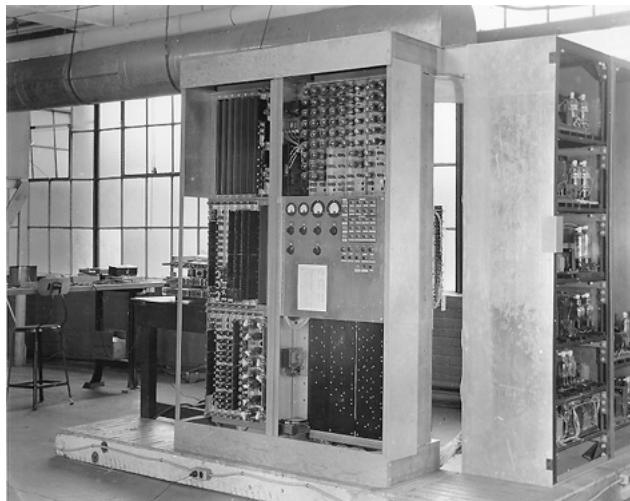


Figura 1.9: Computadora EDVAC.

La idea fundamental de *Von Neumann* fue: permitir que en la memoria coexistan datos con instrucciones, para que entonces la computadora pueda ser programada en un lenguaje, y no por medio de alambres que eléctricamente interconectaban varias secciones de control, como en la ENIAC.

Todo este desarrollo de las computadoras suele divisarse por generaciones y el criterio que se determinó para determinar el cambio de generación no está muy bien definido, pero resulta aparente que deben cumplirse al menos los siguientes requisitos:

La forma en que están construidas.

Forma en que el ser humano se comunica con ellas.



Figura 1.10: von Neumann en 1940.

### **1.2.1. Primera Generación**

En esta generación había una gran desconocimiento de las capacidades de las computadoras, puesto que se realizó un estudio en esta época que determinó que con veinte computadoras se saturaría el mercado de los Estados Unidos en el campo de procesamiento de datos.

Esta generación abarco la década de los cincuenta. Y se conoce como la primera generación. Estas máquinas tenían las siguientes características:

Estas máquinas estaban construidas por medio de tubos de vacío.

Eran programadas en lenguaje de máquina. En esta generación las máquinas son grandes y costosas (de un costo aproximado de ciento de miles de dólares).

En 1951 aparece la UNIVAC (UNIVersAl Computer), fue la primera computadora comercial, que disponía de mil palabras de memoria central y podían leer cintas magnéticas, se utilizó para procesar el censo de 1950 en los Estados Unidos.

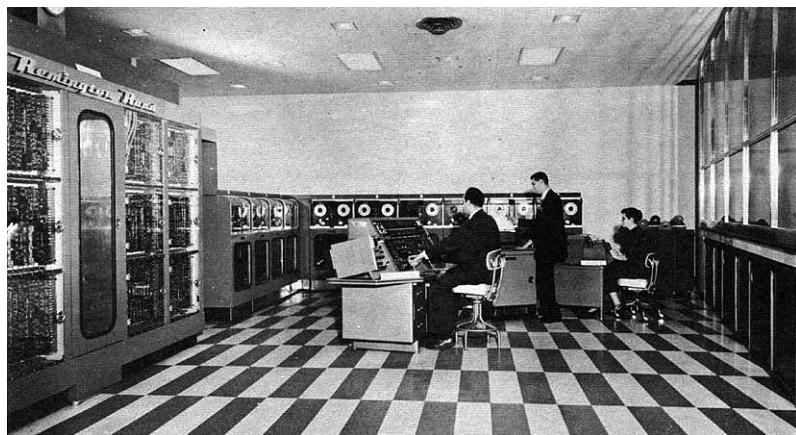


Figura 1.11: UNIVAC I en Franklin Life Insurance Company.

En las dos primeras generaciones, las unidades de entrada utilizaban tarjetas perforadas, retomadas por Herman Hollerith (1860 - 1929), quien además fundó una compañía que con el paso del tiempo se conocería como IBM (International Business Machines).

Después se desarrolló por IBM la IBM 701 de la cual se entregaron 18 unidades entre 1953 y 1957.

Posteriormente, la compañía Remington Rand fabricó el modelo 1103, que competía con la 701 en el campo científico, por lo que la IBM desarrolló

la 702, la cual presentó problemas en memoria, debido a esto no duró en el mercado.

La computadora más exitosa de la primera generación fue la IBM 650, de la cual se produjeron varios cientos. Esta computadora que usaba un esquema de memoria secundaria llamado tambor magnético, que es el antecesor de los discos actuales. Otros modelos de computadora que se pueden situar en los inicios de la segunda generación son: la UNIVAC 80 y 90, las IBM 704 y 709, Burroughs 220 y UNIVAC 1105.



Figura 1.12: IBM 650.

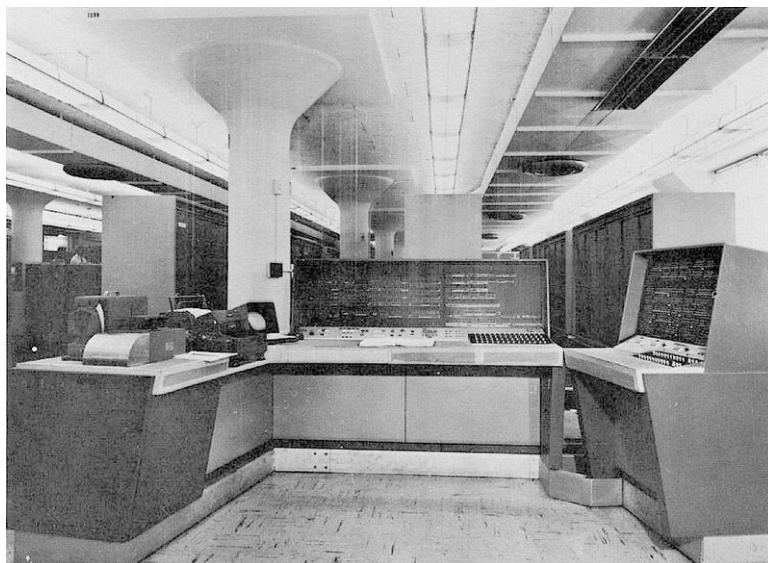


Figura 1.13: IUNIVAC 1105 operator console, in front of the cabinets containing the CPU and memory.

### 1.2.2. Segunda Generación

Cerca de la década de 1960, las computadoras seguían evolucionando, se reducía su tamaño y crecía su capacidad de procesamiento. También en esta época se empezó a definir la forma de comunicarse con las computadoras, que recibía el nombre de programación de sistemas.

Las características de la segunda generación son las siguientes: Están construidas con circuitos de transistores. Se programan en nuevos lenguajes llamados lenguajes de alto nivel.

En esta generación las computadoras se reducen de tamaño y son de menor costo. Aparecen muchas compañías y las computadoras eran bastante avanzadas para su época como la serie 5000 de Burroughs y la ATLAS de la Universidad de Manchester.



Figura 1.14: The University of Manchester Atlas in January 1963.

Algunas de estas computadoras se programaban con cintas perforadas y otras más por medio de cableado en un tablero. Los programas eran hechos a la medida por un equipo de expertos: analistas, diseñadores, programadores y operadores que se manejaban como una orquesta para resolver los problemas y cálculos solicitados por la administración. El usuario final de la información no tenía contacto directo con las computadoras. Esta situación en un principio se produjo en las primeras computadoras personales, pues se requería saberlas “programar” (alimentarle instrucciones) para obtener resultados; por lo tanto su uso estaba limitado a aquellos audaces pioneros que gustaran de pasar un buen número de horas escribiendo instrucciones, “corriendo” el programa resultante y verificando y corrigiendo los errores o bugs que aparecieran. Además, para no perder el “programa” resultante había que “guardarlo” (almacenarlo) en una grabadora de astte, pues en esa época no había discos flexibles y mucho menos discos duros para las PC; este procedimiento podía tomar de 10 a 45 minutos, según el programa. El panorama se modificó totalmente con la aparición de las computadoras personales con mejore circuitos, más memoria, unidades de disco flexible y sobre todo con la aparición de programas de aplicación general en donde el usuario compra el programa y se pone a trabajar. Aparecen los programas procesadores de palabras como el célebre Word Star, la impresionante hoja de cálculo (spreadsheet) Visicalc y otros más que de la noche a la mañana cambian la imagen de la PC. El software empieza a tratar de alcanzar el paso del hardware. Pero aquí aparece un nuevo elemento: el usuario.



Figura 1.15: WordStar corriendo en DOS.

El usuario de las computadoras va cambiando y evolucionando con el tiempo. De estar totalmente desconectado a ellas en las máquinas grandes pasa la PC a ser pieza clave en el diseño tanto del hardware como del software. Aparece el concepto de human interface que es la relación entre el usuario y su computadora. Se habla entonces de hardware ergonómico (adaptado a las dimensiones humanas para reducir el cansancio), diseños de pantallas anti-reflejos y teclados que descansen la muñeca. Con respecto al software se inicia una verdadera carrera para encontrar la manera en que el usuario pase menos tiempo capacitándose y entrenándose y más tiempo produciendo. Se ponen al alcance programas con menús (listas de opciones) que orientan en todo momento al usuario (con el consiguiente aburrimiento de los usuarios expertos); otros programas ofrecen toda una artillería de teclas de control y teclas de funciones (atajos) para efectuar toda suerte de efectos en el trabajo (con la consiguiente desorientación de los usuarios novatos). Se ofrecen un sin número de cursos prometiendo que en pocas semanas hacen de cualquier persona un experto en los programas comerciales. Pero el problema “constante” es que ninguna solución para el uso de los programas es “constante”. Cada nuevo programa requiere aprender nuevos controles, nuevos trucos, nuevos menús. Se empieza a sentir que la relación usuario-PC no está acorde con los desarrollos del equipo y de la potencia de los programas. Hace falta una relación amistosa entre el usuario y la PC.

Las computadoras de esta generación fueron: la Philco 212 (esta compañía se retiró del mercado en 1964) y la UNIVAC M460, la Control Data

Corporation modelo 1604, seguida por la serie 3000, la IBM mejoró la 709 y sacó al mercado la 7090, la National Cash Register empezó a producir máquinas para proceso de datos de tipo comercial, introdujo el modelo NCR 315. La Radio Corporation of America introdujo el modelo 501, que manejaba el lenguaje COBOL, para procesos administrativos y comerciales. Después salió al mercado la RCA 601.

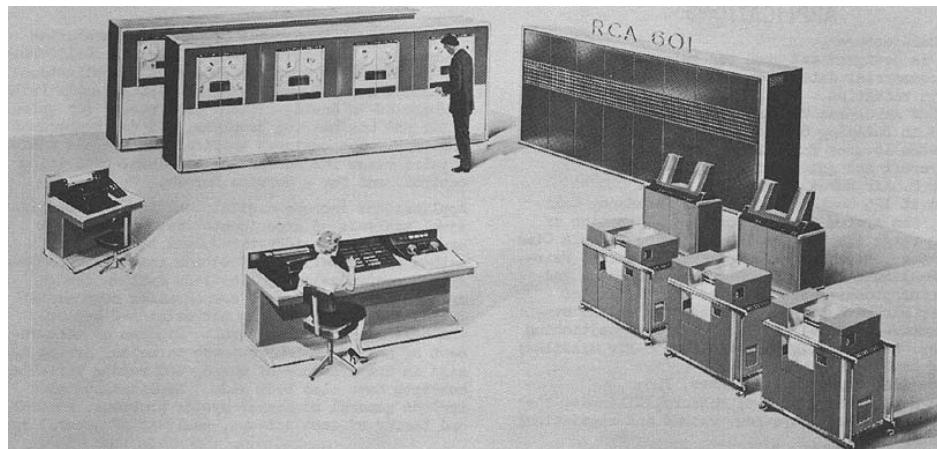


Figura 1.16: RCA-601.

### 1.2.3. Tercera generación

Con los progresos de la electrónica y los avances de comunicación con las computadoras en la década de los 1960, surge la tercera generación de las computadoras. Se inaugura con la IBM 360 en abril de 1964.



Figura 1.17: IBM-360.

Las características de esta generación fueron las siguientes:

- Su fabricación electrónica esta basada en circuitos integrados.
- Su manejo es por medio de los lenguajes de control de los sistemas operativos.

La IBM produce la serie 360 con los modelos 20, 22, 30, 40, 50, 65, 67, 75, 85, 90, 195 que utilizaban técnicas especiales del procesador, unidades de cinta de nueve canales, paquetes de discos magnéticos y otras características que ahora son estándares (no todos los modelos usaban estas técnicas, sino que estaba dividido por aplicaciones). El sistema operativo de la serie 360, se llamó OS que contaba con varias configuraciones, incluía un conjunto de técnicas de manejo de memoria y del procesador que pronto se convirtieron en estándares.

En 1964 CDC introdujo la serie 6000 con la computadora 6600 que se consideró durante algunos años como la más rápida. En la década de 1970, la IBM produce la serie 370 (modelos 115, 125, 135, 145, 158, 168). UNIVAC

compite son los modelos 1108 y 1110, máquinas en gran escala; mientras que CDC produce su serie 7000 con el modelo 7600. Estas computadoras se caracterizan por ser muy potentes y veloces.

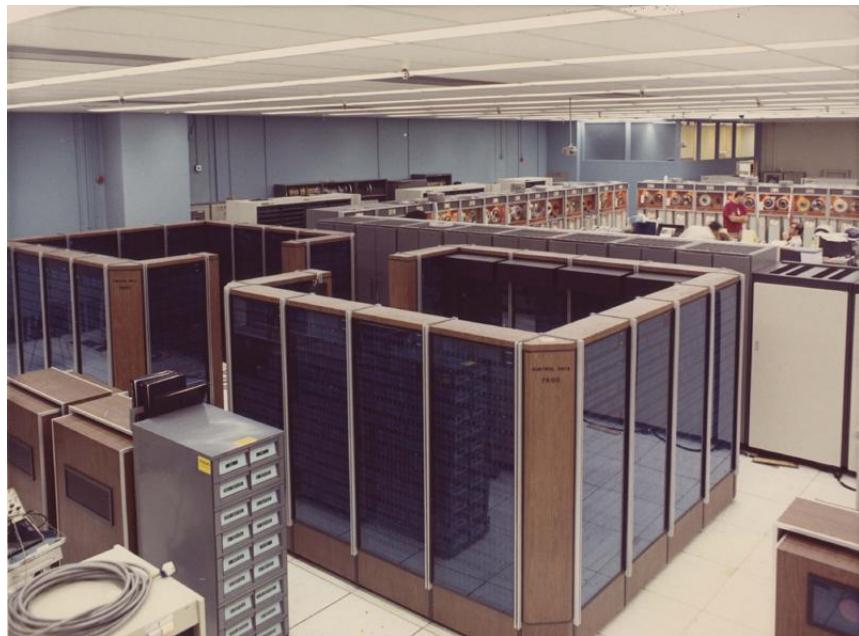


Figura 1.18: CDC-7600.

A finales de esta década la IBM de su serie 370 produce los modelos 3031, 3033, 4341. Burroughs con su serie 6000 produce los modelos 6500 y 6700 de avanzado diseño, que se reemplazaron por su serie 7000. Honey - Well participa con su computadora DPS con varios modelos.



Figura 1.19: System /370-145.

A mediados de la década de 1970, aparecen en el mercado las computadoras de tamaño mediano, o minicomputadoras que no son tan costosas como las grandes (llamadas también como mainframes que significa también, gran sistema), pero disponen de gran capacidad de procesamiento. Algunas minicomputadoras fueron las siguientes: la PDP - 8 y la PDP - 11 de Digital Equipment Corporation, la VAX (Virtual Address eXtended) de la misma compañía, los modelos NOVA y ECLIPSE de Data General, la serie 3000 y 9000 de Hewlett - Packard con varios modelos el 36 y el 34, la Wang y Honey - Well -Bull, Siemens de origen alemán, la ICL fabricada en Inglaterra. En la Unión Soviética se utilizó la US (Sistema Unificado, Ryad) que ha pasado por varias generaciones.

#### 1.2.4. Cuarta Generación

Aquí aparecen los microprocesadores que es un gran adelanto de la microelectrónica, son circuitos integrados de alta densidad y con una velocidad impresionante. Las microcomputadoras con base en estos circuitos son ex-

tremadamente pequeñas y baratas, por lo que su uso se extiende al mercado industrial. Aquí nacen las computadoras personales que han adquirido proporciones enormes y que han influido en la sociedad en general sobre la llamada “revolución informática”.

En 1976 Steve Wozniak y Steve Jobs inventan la primera microcomputadora de uso masivo y más tarde forman la compañía conocida como la Apple que fue la segunda compañía más grande del mundo, antecedida tan solo por IBM; y esta por su parte es aún de las cinco compañías más grandes del mundo. En 1981 se vendieron 800 00 computadoras personales, al siguiente subió a 1 400 000. Entre 1984 y 1987 se vendieron alrededor de 60 millones de computadoras personales, por lo que no queda duda que su impacto y penetración han sido enormes.



Figura 1.20: Steve Jobs y Steve Wozniak.

Con el surgimiento de las computadoras personales, el software y los sistemas que con ellas manejan han tenido un considerable avance, porque han hecho más interactiva la comunicación con el usuario. Surgen otras aplicaciones como los procesadores de palabra, las hojas electrónicas de cálculo, paquetes gráficos, etc. También las industrias del Software de las computadoras personales crece con gran rapidez, Gary Kildall y William Gates se dedicaron durante años a la creación de sistemas operativos y métodos para lograr una utilización sencilla de las microcomputadoras (son los creadores de CP/M y de los productos de Microsoft).

No todo son microcomputadoras, por su puesto, las minicomputadoras y los grandes sistemas continúan en desarrollo. De hecho las máquinas pequeñas rebasaban por mucho la capacidad de los grandes sistemas de 10 o 15 años antes, que requerían de instalaciones costosas y especiales, pero sería equivocado suponer que las grandes computadoras han desaparecido; por el contrario, su presencia era ya ineludible en prácticamente todas las esferas de control gubernamental, militar y de la gran industria.

Las enormes computadoras de las series CDC, CRAY, Hitachi o IBM por ejemplo, eran capaces de atender a varios cientos de millones de operaciones por segundo.

### 1.2.5. Quinta Generación

En vista de la acelerada marcha de la microelectrónica, la sociedad industrial se ha dado a la tarea de poner también a esa altura el desarrollo del software y los sistemas con que se manejan las computadoras. Surge la competencia internacional por el dominio del mercado de la computación, en la que se perfilan dos líderes que, sin embargo, no han podido alcanzar el nivel que se desea: la capacidad de comunicarse con la computadora en un lenguaje más cotidiano y no a través de códigos o lenguajes de control especializados.

Japón lanzó en 1983 el llamado “programa de la quinta generación de computadoras”, con los objetivos explícitos de producir máquinas con innovaciones reales en los criterios mencionados. Y en los Estados Unidos ya está en actividad un programa en desarrollo que persigue objetivos semejantes, que pueden resumirse de la siguiente manera: Procesamiento en paralelo mediante arquitecturas y diseños especiales y circuitos de gran velocidad. Manejo de lenguaje natural y sistemas de inteligencia artificial.

El futuro previsible de la computación es muy interesante, y se puede esperar que esta ciencia siga siendo objeto de atención prioritaria de gobiernos y de la sociedad en conjunto.



# Capítulo 2

## Lenguajes y Gramáticas

### 2.1. Gödel y Turing

En el año 1931 se produjo una revolución en las ciencias matemáticas, con el descubrimiento realizado *Kurt Gödel* (1906-1978), que quizá deba considerarse el avance matemático más importante del siglo XX : Toda formulación axiomática de la teoría de números contiene proposiciones indecibles. Es decir cualquier teoría matemática ha de ser incompleta. Siempre habrá en ella afirmaciones que no se podrán demostrar ni negar.

El teorema de *Gödel* puso punto final a las esperanzas de los matemáticos de construir un sistema completo y consistente, en el que fuese posible demostrar cualquier teorema. Estas esperanzas habían sido expresadas en 1900 *David Hilbert* (1862-1943), quien generalizó sus puntos de vista proponiendo el problema de la decisión, cuyo objetivo era descubrir un método general para decidir si una fórmula lógica es verdadera o falsa.

En 1937, el matemático inglés *Alan Mathison Turing* (1912-1953) publicó otro artículo famoso sobre los números calculables, que desarrolló el teorema de *Gödel* y puede considerarse el origen oficial de la computación. En este

artículo introdujo la máquina de Turing, una entidad matemática abstracta que formalizó por primera vez el concepto de *algoritmo* y resultó ser precursora de las máquinas de calcular automáticas, que comenzaron a extenderse a partir de la década siguiente. Además el teorema de Turing demostraba que existen problemas irresolubles, es decir, que ninguna máquina de Turing, y por ende ninguna computadora, será capaz de obtener su solución?. Por ello se considera a Turing como el padre de la teoría de la computabilidad.

El *teorema de Turing* es, en el fondo, equivalente al teorema de Gödel. Si el segundo demuestra que no todos los teoremas pueden demostrarse, el primero dice que no todos los problemas pueden resolverse. Además la demostración de ambos teoremas es muy parecida y son realizadas por reducción al absurdo.

## 2.2. Autómatas

En 1938, otro artículo famoso del matemático norteamericano *Claude Elwood Shannon (1916-2001)*, vino a establecer las bases para la aplicación de la lógica matemática a los circuitos combinatorios y secuenciales, construidos al principio con relés y luego con dispositivos electrónicos de vacío y estado sólido. A lo largo de las décadas siguientes, las ideas de Shannon se convirtieron en la teoría de las máquinas secuenciales y de los autómatas finitos.

Los autómatas son sistemas capaces de transmitir información. En sentido amplio, todo sistema que acepta señales de su entorno y, como resultado, cambia de estado y transmite otras señales al medio, puede considerarse como un autómata. Con esta definición, cualquier máquina, un teléfono, una computadora e incluso los seres vivos, los seres humanos y las sociedades se comportarían como autómatas. Este concepto de autómata es demasiado general para su estudio teórico, por lo que se introducen limitaciones en su definición.

Desde su nacimiento, la teoría de autómatas encontró aplicación en campos muy diversos, pero que tienen en común el manejo de conceptos como

control, la acción, la memoria.

### 2.3. Lenguajes y Gramáticas

En la década de 1950, el lingüista norteamericano *Avram Noam Chomsky* (1928-) revolucionó su campo de actividad con la teoría de las gramáticas transformacionales, que estableció las bases de la lingüística matemática y proporcionó una herramienta que, aunque Chomsky la desarrolló para aplicarla a los lenguajes naturales, facilitó considerablemente el estudio y la formalización de los lenguajes de computadora, que comenzaban a aparecer precisamente en aquella época.

El estudio de los lenguajes se divide en el análisis de la estructura de las frases (gramática) y de su significado (semántica). A su vez, la gramática puede analizar las formas que toman las palabras (morfología), su combinación para formar frases correctas (sintaxis) y las propiedades del lenguaje hablado (fonética). Por ahora, tan sólo esta última no se aplica a los lenguajes de computadora.

Aunque desde el punto de vista teórico la distinción entre sintaxis y semántica es un poco artificial, tiene una enorme trascendencia desde el punto de vista práctico, especialmente para diseño y construcción de compiladores.

### 2.4. Máquinas Abstractas y Lenguajes Formales

La teoría de los lenguajes formales resultó tener una relación sorprendente con la teoría de máquinas abstractas. Los mismos fenómenos aparecen independientemente en ambas disciplinas y es posible establecer un isomorfismo entre ellas.

Chomsky clasificó las gramáticas y los lenguajes formales de acuerdo con una jerarquía de cuatro grados, cada uno de los cuales contiene a todos los siguientes.

Pues bien : paralelamente a esta jerarquía de gramáticas y lenguajes, existe otra de máquinas abstractas equivalentes. La tabla siguiente resume la relación entre las cuatro jerarquías de las gramáticas, los lenguajes, las máquinas abstractas y los problemas que son capaces de resolver.

			Problemas no computables
Gramáticas tipo 0	Lenguajes computables	Máquinas de Turing	Problemas recursivamente enumerables
Gramáticas tipo 1	Lenguajes dependientes del contexto	Autómatas Lineales Acotados	
Gramáticas tipo 2	Lenguajes independientes del contexto	Autómatas de Pila	
Gramáticas tipo 3	Lenguajes regulares	Autómatas finitos	Expresiones regulares

## 2.5. Alfabetos, Símbolos y palabras

Se llama alfabeto a un conjunto finito, no vacío. Los elementos del alfabeto se llaman símbolos. Por ejemplo:

### Ejemplo 2.5.1.

$$\Sigma_1 = \{a, b, c, d, e, \dots, z\}$$

$$\Sigma_2 = \{0, 1\}$$

$$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$\Sigma_4$  = Todas las palabras del diccionario

Se llama palabra (cadena , oración) a una secuencia finita de símbolos de un alfabeto. Por ejemplo:

**Ejemplo 2.5.2.**

$s = \text{computacion}$  ( palabra sobre  $\Sigma_1$ )  
 $u = 110010111$  (palabra sobre  $\Sigma_2$ )  
 $v = 5047850$  (palabra sobre  $\Sigma_3$ )  
 $w = \text{en computación el equivalente al novel,}$   
 es la medalla turing (oración sobre  $\Sigma_4$ )

Se llama longitud de una palabra al número de símbolos que la componen. La palabra cuya longitud es cero se llama palabra vacía y se representa con  $\lambda$ . Por ejemplo:

**Ejemplo 2.5.3.**

$$\begin{aligned} |w| &= 11 \\ |v| &= 9 \\ |u| &= 7 \\ |\lambda| &= 0 \end{aligned}$$

El conjunto de todas las palabras que se pueden formar con los símbolos de un alfabeto se llama lenguaje universal de  $\Sigma$ . De momento se utilizará la notación  $W(\Sigma)$  para representarlo. Es evidente que  $W(\Sigma)$  es un conjunto infinito. Por ejemplo:

$$\begin{aligned} W(\Sigma_1) &= \{\lambda, \text{informatica}, \text{aabbbcc}, \text{a}, \text{b}, \text{logica}, \text{sxyw}, \dots\} \\ W(\Sigma_2) &= \{\lambda, 0, 1, 001, 11, 1110101, \dots\} \\ W(\{\text{a}\}) &= \{\lambda, \text{a}, \text{aa}, \text{aaa}, \text{aaaa}, \dots\} \end{aligned}$$

Obsérvese que la palabra vacía pertenece a todos los lenguajes universales de todos los alfabetos posibles.

## 2.6. Operaciones con Palabras

La concatenación de palabras  $u$  con  $v$ , es la palabra  $w = uv$  que se obtiene poniendo los símbolos de  $u$  y a continuación los símbolos de  $v$ . Por ejemplo:

**Ejemplo 2.6.1.** Si

$$\begin{aligned} u &= a_1, a_2, \dots, a_i \\ v &= b_1, b_2, \dots, b_j \end{aligned}$$

son dos palabras en  $W(\Sigma)$ .

Se tiene que:

$$w = uv = a_1, a_2, \dots, a_i, b_1, b_2, \dots, b_j$$

La concatenación también se puede representar con  $w = u.v$ . Esta operación tiene las siguientes propiedades:

### 2.6.1. Operación Cerrada:

$$u, v \in W \Rightarrow u.v \in W(\Sigma)$$

### 2.6.2. Propiedad Asociativa:

$$u.(v.w) = (u.v).w, \forall u, v, w \in W(\Sigma)$$

Por cumplir las dos propiedades anteriores:  $(W(\Sigma), .)$  es un semigrupo.

### 2.6.3. Existencia de Elemento Neutro:

$$\exists \lambda \in W(\Sigma) / \lambda.u = u.\lambda, \forall u \in W(\Sigma)$$

Por cumplir las dos propiedades anteriores:  $(W(\Sigma), .)$  es un monoide.

### 2.6.4. Potencia de una Palabra:

Puesto que  $(W(\Sigma), .)$  es un monoide, en él se define la potencia de una palabra como sigue:

$$\begin{aligned} u^k &= u.u^{k-1} & , k > 0 \\ u^0 &= \lambda \end{aligned}$$

### 2.6.5. Reflexión de una Palabra:

Sea  $u = a_1, a_2, \dots, a_n$ . Se llama palabra refleja o inversa de  $u$ , a la palabra  $u^{-1} = a_n, \dots, a_2, a_1$

## 2.7. Lenguajes

Todo subconjunto  $L$  del lenguaje universal  $W(\Sigma)$  diremos que es un lenguaje sobre  $\Sigma$ . En particular, el conjunto vacío  $\Phi$  y  $W(\Sigma)$  son lenguajes sobre  $\Sigma$ . Como los lenguajes son conjuntos es claro que sobre los lenguajes se definen todas las operaciones clásicas de los conjuntos y que en ellos también conservan las propiedades de la teoría de conjuntos.

### 2.7.1. Concatenación de Lenguajes

La concatenación de los lenguajes  $L_1 \subset W(\Sigma)$  y  $L_2 \subset W(\Sigma)$  es el lenguaje:

$$L_1.L_2 = \{u.v / u \in L_1 \wedge v \in L_2\}$$

La concatenación de lenguajes tienen las siguientes propiedades:

1.  $\Phi.L = L.\Phi = \Phi$

2. Operación cerrada:

$$L_1 \subset W(\Sigma) \wedge L_2 \subset W(\Sigma) \Rightarrow L_1.L_2 \subset W(\Sigma)$$

3. Propiedad asociativa:

$$L_1.(L_2.L_3) = (L_1.L_2).L_3$$

4. Existencia de elemento neutro:

$$\{\lambda\}.L = L.\{\lambda\} = L$$

Por cumplir las tres propiedades anteriores ( $P(W(\Sigma))$ , .) es un monoides. Por tanto en  $P(W(\Sigma))$  se define la potencia de lenguajes :

$$\begin{aligned} L^k &= L.L^{k-1} & , k > 0 \\ L^0 &= \{\lambda\} \end{aligned}$$

### 2.7.2. Clausura Positiva de un Lenguaje

Es el lenguaje  $L^+$ , obtenido uniendo el lenguaje  $L$ , con todas sus potencias posibles, excepto  $L^0$ . Es decir,  $L^+ = \bigcup_{i=1}^{\infty} L^i$ . Obviamente, ninguna clausura positiva contiene la palabra vacía, a menos que dicha palabra está en  $L$ .

Puesto que el alfabeto  $\Sigma$  es también un lenguaje sobre  $\Sigma$ , pueden aplicársele esta operación, entonces  $\Sigma^* = W(\Sigma) - \{\lambda\}$ .

### 2.7.3. Iteración, Cierre o Clausura de un Lenguaje

Es el lenguaje  $L^*$ , obtenido uniendo el lenguaje  $L$ , con todas sus potencias posibles, incluso  $L^0$ . Es decir,  $L^* = \bigcup_{i=0}^{\infty} L^i$ . Son evidentes las siguientes identidades:

1.  $L^* = L^+ \bigcup \{\lambda\}$
2.  $L^+ = L \cdot L^* = L^* \cdot L$
3.  $\Sigma^* = W(\Sigma)$

Que es la notación que utilizaremos a partir de este momento.

4. Para todo alfabeto  $\Sigma$ ,  $\Sigma^*$  es infinito numerable.
5. El conjunto de todos lenguajes sobre  $\Sigma$  no es numerable.

### 2.7.4. Reflexión de Lenguajes

Se llama lenguaje reflejo o inverso de  $L$  al lenguaje

$$L^{-1} = \{u^{-1} / u \in L\}$$

## 2.8. Conceptos Básicos sobre Gramáticas

En primer lugar, debemos esclarecer los términos lenguaje formal y lenguaje natural. Es posible distinguir a ambos por medio de la pregunta, que

surgió primero, el lenguaje o sus reglas gramaticales?. En general, un lenguaje natural es aquel que ha evolucionado con el paso del tiempo para fines de comunicación humana, por ejemplo el inglés, el español o el alemán. Estos lenguajes continúan su evolución sin tomar en cuenta reglas gramaticales formales; cualquier regla se desarrolla después de que sucede el hecho, en un intento por explicar, y no determinar, la estructura del lenguaje. Como resultado de esto, pocas veces los lenguajes naturales se ajustan a reglas gramaticales sencillas. Esta situación es el pelo en la sopa de los académicos de la lengua, la mayoría de las cuales trata de convencernos de que nuestra forma de hablar es incorrecta en vez de aceptar la posibilidad de que no hayan encontrado el conjunto de reglas adecuado, que sus reglas ya no son válidas o que no existen dichas reglas.

En contraste con los lenguajes naturales, los formales están definidos por reglas preestablecidas y, por lo tanto, se ajustan con todo rigor a ellas. Como ejemplo tenemos los lenguajes de programación y la matemática. Gracias a esta adhesión a las reglas, es posible construir traductores computarizados eficientes para los lenguajes de programación, a la vez que la falta de adhesión a las reglas establecidas dificulta la construcción de un traductor para un lenguaje natural.

Las gramáticas formales son descripciones estructurales de las sentencias de los lenguajes, tanto formales, como naturales. Las definiciones proporcionadas anteriormente de los lenguajes son extensionales; es decir, para describir el lenguaje se enumeran todas las palabras que lo forman. Las gramáticas formales que pasaremos a definir permiten describir por comprensión a los lenguajes.

### 2.8.1. Gramática de Estructura de Frases

Se llama Gramática de Estructura de Frases a la cuádrupla

$$G = (\Sigma_T, \Sigma_N, v_0, \vdash \rightarrow)$$

donde:

- $\Sigma_T$  es un alfabeto llamado de símbolos terminales.
- $\Sigma_N$  es un alfabeto llamado de símbolos no terminales. Se verifica que  $\Sigma = \Sigma_T \cup \Sigma_N$  y  $\Sigma_T \cap \Sigma_N = \emptyset$ .
- $v_0 \in \Sigma_N$  es el símbolo inicial ó axioma.
- $\mapsto$  es una relación finita en  $\Sigma^*$ , sus elementos  $u \mapsto v$  se llaman producciones y especifica los reemplazos permisibles, en el sentido de que se puede reemplazar  $u$  por  $v$ .

También es posible introducir la relación  $\Rightarrow$  en  $\Sigma^*$  llamada de derivabilidad directa ;  $x \Rightarrow y$  significa que  $x = l.u.r$ ,  $y = l.v.r$  siempre y cuando  $u \mapsto v$  y diremos que  $y$  se deriva directamente de  $x$ .

Se decreta que una cadena  $w \in \Sigma_T^*$  es una oración sintácticamente correcta si y sólo si  $v_0 \Rightarrow^\infty w$

El lenguaje generado por  $G$  o el lenguaje de  $G$ , denotado por  $L(G)$  es el conjunto de todas las cadenas que son sintácticamente correctas en  $G$ , es decir :

$$L(G) = \{w \in \Sigma_T^* / v_0 \Rightarrow^\infty w\}$$

**Ejemplo 2.8.1.** Sea:

$\Sigma_T = \{\text{la, soñolienta, feliz, tortuga, liebre, rebasa, a, corre, rápidamente, lentamente}\}$   
 $\Sigma_N = \{\text{oración, sujeto, predicado.con.verbo.transitivo, predicado.verbo.intransitivo, artículo, adjetivo, nombre, verbo, adverbio}\}$

y las producciones :

oración  $\mapsto$  sujeto, predicado.con.verbo.transitivo, sujeto

oración  $\mapsto$  sujeto, predicado.con.verbo.intransitivo

sujeto  $\mapsto$  artículo, nombre, adjetivo

sujeto  $\mapsto$  artículo, nombre predicado.con.verbo.transitivo  $\mapsto$  verbo.transitivo

predicado.con.verbo.intransitivo  $\mapsto$  verbo.intransitivo, adverbio

predicado.con.verbo.intransitivo  $\mapsto$  verbo.intransitivo

artículo  $\mapsto$  la

adjetivo  $\mapsto$  feliz

adjetivo  $\mapsto$  soñolienta

nombre  $\mapsto$  tortuga

nombre  $\mapsto$  liebre

verbo.transitivo  $\mapsto$  rebasa, a

verbo.intransitivo  $\mapsto$  corre

adverbio  $\mapsto$  rápidamente

adverbio  $\mapsto$  lentamente

entonces;  $G = (\Sigma_T, \Sigma_N, \text{oración}, \mapsto)$  es una gramática de estructura de frase.

Las siguientes oraciones pertenecen a  $L(G)$ ?

- la liebre feliz corre
- la tortuga soñolienta corre rápidamente.
- la liebre corre a la tortuga soñolienta
- la liebre soñolienta rebasa a la tortuga feliz
- la tortuga rebasa a la liebre

**Observación 2.8.1.** Se debe tener en cuenta que

- a. Sean  $v, w$  dos palabras del mismo alfabeto. Se dice que existe una relación  $\text{Tue}$  entre  $v, w$  si se verifica que  $v \mapsto w^\infty v = w$ .
- b. Una gramática  $G$  se llama recursiva en  $v$  si  $v \mapsto xvy$ . Si  $x = \lambda$  se dice que la gramática es recursiva a izquierdas. Si  $y = \lambda$  se dice que la gramática es recursiva a derechas. Si un lenguaje es infinito, la gramática que lo genera tiene que ser recursiva.

## 2.8.2. Tipos de Gramáticas

Chomsky clasificó a las gramáticas en cuatro grandes grupos, cada uno de los cuales incluye a los siguientes.

### 1. Gramáticas de Tipo 0:

Son las más generales. Las reglas de producción no tienen ninguna restricción. Los lenguajes generados por estas gramáticas se llaman lenguajes sin restricciones.

### 2. Gramáticas de Tipo 1:

Las reglas de producción de estas gramáticas  $v \mapsto w$  son tales que  $|v| \leq |w|$ . Los lenguajes representados por este tipo de gramáticas se llaman dependientes del contexto o sensibles al contexto.

3. Gramáticas del Tipo 2:

Las reglas de producción de estas gramáticas tienen la forma  $v \mapsto w$  donde  $v \in \Sigma_N$ ,  $w \in \Sigma^*$ . Los lenguajes representados por este tipo de gramáticas se llaman independientes del contexto, pues la conversión de  $v$  en  $w$  puede aplicarse cualquiera que sea el contexto donde se encuentre  $v$ . La sintaxis de casi todos los lenguajes de programación pueden describirse mediante este tipo de gramáticas.

4. Gramáticas del Tipo 3:

Las producciones  $v \mapsto w$  de estas gramáticas verifican las condiciones de las gramáticas del tipo 2 y además deberán cumplir que  $w$  tenga a lo más un símbolo de  $\Sigma_N$  y éste debe estar más a la derecha. Los lenguajes generados por este tipo de gramáticas se denominan lenguajes regulares.

**Observación 2.8.2.** Tenga en claro que

- a. Las gramáticas, que resultan ser modelos muy tiles para el desarrollo de software destinado a procesar datos que posean estructura recursiva. El ejemplo más conocido es el analizador sintáctico, ó parecer, que es la componente del compilador que se ocupa de las características recursivas de los lenguajes de programación típicos, como las expresiones (aritméticas, condicionales, etc.)
- b. Las gramáticas del Tipo 2 tienen métodos alternativos tiles para desplegar las producciones.
- c. Una notación alternativa es la notación BNF ( forma Backus-Naur). Los símbolos no terminales se encierran entre  $|$ ,  $\{$  y todos los lados derechos correspondientes a un símbolo no terminal del lado izquierdo son enumerados juntos, separados por el símbolo  $\rightarrow$ . El símbolo  $\mapsto$  se reemplaza por el símbolo  $::=$ .
- d. Un segundo método alternativo para desplegar las producciones de las gramáticas del Tipo 2 son los diagramas sintácticos. ésta es una imagen de cada una de las producciones que permite al usuario ver las sustituciones en forma din?ica. La junta de todos los diagramas sintácticos donde sólo aparece el símbolo inicial  $v_0$  se llama diagrama maestro.

**Ejemplo 2.8.2.** Describa el lenguaje generado por la siguiente gramática

```

< numero – decimal > ::= < entero – sin – signo > | < decimal > |
    < entero – sin – signo >< decimal >
    < decimal > ::= < entero – sin – signo >
< entero – si – signo > ::= < digito > | < digito >< entero – sin – signo >
    < digito > ::= 0|1|2|3|4|5|6|7|8|9
  
```

**Ejemplo 2.8.3.** Dibujar los diagramas sintácticos y el diagrama maestro de la gramática del ejemplo anterior.

## 2.9. Expresiones Regulares

Las expresiones regulares pueden describirse recursivamente de la siguiente forma. En esta definición no sólo describimos que son expresiones regulares, sino que para cada expresión regular  $\alpha$  describiremos el lenguaje que representa, al que llamaremos  $L(\alpha)$ .

1. El símbolo  $\Phi$  (conjunto vacío) es una expresión regular y  $L(\Phi) = \{\}$
2.  $\lambda$  es una expresión regular y  $L(\lambda) = \{\lambda\}$ .
3. Si  $a \in \Sigma$  entonces el símbolo  $a$  es una expresión regular y  $L(a) = \{a\}$ .
4. Si  $\alpha, \beta$  son expresiones regulares, entonces  $\alpha \vee \beta$ ,  $\alpha \cdot \beta$ ,  $\alpha^*$  también son expresiones regulares y  $L(\alpha \vee \beta) = L(\alpha) \cup L(\beta)$ ,  $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$   $L(\alpha^*) = (L(\alpha))^*$

**Ejemplo 2.9.1.** Las siguientes son expresiones regulares en  $\Sigma = \{0, 1\}$  junto con el lenguaje que le corresponde:

1.  $L(1) = \{1\}$
2.  $L((0 \vee 11)01) = \{001, 1101\}$

$$3. L(1^*00) = \{00, 100, 1100, 11100, 111100, \dots\}$$

**Definición 2.1.** Sean  $r, s, t$  expresiones regulares sobre el mismo alfabeto  $\Sigma$ . entonces :

1.  $r \vee s = s \vee r$
2.  $r \vee \Phi = r = \Phi \vee r$
3.  $r \vee r = r$
4.  $r \vee (s \vee t) = (r \vee s) \vee t$
5.  $r.\lambda = \lambda r = r$
6.  $r.\Phi = \Phi r = \Phi$
7.  $(r.s).t = r.(s.t)$
8.  $r.(s \vee t) = r.s \vee r.t, (s \vee t).r = s.r \vee t.r$
9.  $r^* = r^{**} = r^*r^* = (\lambda \vee r)^* = r^*. (r \vee \lambda) = (r \vee \lambda).r^* = \lambda \vee r.r^*$
10.  $(r \vee s)^* = (r^* \vee s^*)^* = (r^*s^*)^*$
11.  $r.(s.r)^* = (r.s)^*.r$
12.  $(r^*.s)^* = \lambda \vee (r \vee s)^*s$
13.  $(r.s^*)^* = \lambda \vee r.(r \vee s)^*$
14.  $s.(r \vee \lambda)^*(r \vee \lambda) \vee s = s.r^*$
15.  $r.r^* = r^*.r$

**Ejemplo 2.9.2.** Simplificar las siguientes expresiones regulares

1.  $(\lambda \vee aa)^*$
2.  $(\lambda \vee aa).(\lambda \vee aa)^*$
3.  $a.(\lambda \vee aa)^*a \vee \lambda$
4.  $(a \vee \lambda).a^*.b$

## 2.10. Árboles de Derivación

Toda derivación de una gramática de Tipo 1, 2, 3 puede representarse mediante un árbol, que se construye de la siguiente manera:

1. La rama del árbol se denota por el axioma de la gramática.
2. Una derivación directa se representa por un conjunto de ramas que salen de un nodo. Al aplicar una regla, un símbolo de la parte izquierda queda sustituido por la palabra  $x$  de la parte derecha. Por cada uno de los símbolos de  $x$  se dibuja una rama que parte del nodo dado y termina en otro, denotado por dicho símbolo.

Sean dos símbolos  $A$  y  $B$  en la palabra  $x$ . Si  $A$  está la izquierda de  $B$  en  $x$ , entonces la rama que termina en  $A$  se dibujará a la izquierda de la rama que termina en  $B$ .

A lo largo del proceso de construcción del árbol, los nodos finales de cada paso sucesivo leídos de izquierda a derecha, dan la forma sentencial obtenida por la derivación representada por el árbol.

Se llama rama terminal aquella que se dirige hacia un nodo denotado por un símbolo terminal de la gramática. Este nodo se denomina hoja o nodo terminal del árbol. El conjunto de las hojas del árbol, leídas de izquierda a derecha, da la sentencia generada por la derivación.

## 2.11. Lista de ejercicios

1. Describa con precisión el lenguaje  $L(G)$  producido por las gramáticas :
  - a)  $G = (\Sigma_T, \Sigma_N, v_0, \rightarrow), \Sigma_T = \{a, b\}, \Sigma_N = \{v_0, v_1, v_2\}, v_0 \rightarrow aa v_0, v_0 \rightarrow bb v_1, v_1 \rightarrow bb v_1, v_1 \rightarrow b$

- b)  $G = (\Sigma_T, \Sigma_N, v_0, \rightarrow), \Sigma_T = \{0, 1\}, \Sigma_N = \{v_0, v_1\}, v_0 \mapsto av_0, v_0 \mapsto av_1, v_1 \mapsto bv_1, v_1 \mapsto \lambda$
- c)  $G = (\Sigma_T, \Sigma_N, v_0, \rightarrow), \Sigma_T = \{x, y\}, \Sigma_N = \{v_0\}, v_0 \mapsto xv_0y, v_0 \mapsto xy$
- d)  $G = (\Sigma_T, \Sigma_N, v_0, \rightarrow), \Sigma_T = \{a, (,), +\}, \Sigma_N = \{v_0, v_1, v_2\}, v_0 \mapsto (v_0), v_0 \mapsto a + v_1, v_1 \mapsto a + v_2, v_2 \mapsto a + v_2, v_2 \mapsto a$
- e)  $G = (\Sigma_T, \Sigma_N, v_0, \rightarrow), \Sigma_T = \{a, b, c\}, \Sigma_N = \{v_0, v_1, v_2\}, v_0 \mapsto v_0v_1, v_0v_1 \mapsto v_2v_0, v_2v_0 \mapsto ab, v_2 \mapsto a, v_1 \mapsto c$

2. Del problema anterior en cada caso determine el tipo de gramática.
3. Dada la gramática  $G = (\Sigma_T, \Sigma_N, I, \rightarrow)$ , escribir la expresión regular correspondiente a  $L(G)$

$< I > ::= < L > | < L >< W >$   
 $< W > ::= < L >< W > | < D >< W > | L | D$   
 $< L > ::= a | b | c$   
 $< D > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

4. Escribir una expresión regular que describa números de teléfono. Considerar números internacionales, así como el hecho de que diferentes países tienen diferente número de dígitos para los códigos de zona y los números locales.
5. Diseñar una gramática  $G$  tal que  $L(G) = L((00 \vee 01 \vee 10 \vee 11)^*)$

# **Capítulo 3**

## **Autómatas Finitos**

Se piensa en una máquina como en un sistema que puede aceptar una entrada, que podría producir una salida y que tiene un tipo de memoria interna que puede llevar el registro de cierta información acerca de las entradas anteriores. La condición interna completa de la máquina y de toda su memoria, en un instante particular, constituye el estado de la máquina en ese instante. El estado en que la máquina se encuentra en cualquier instante resume su memoria de las entradas pasadas y determina la forma en que debe reaccionar a la entrada posterior.

En este capítulo estudiaremos la clase de máquinas técnicas conocidas como Autómatas Finitos. Aunque su poder es limitado, encontraremos que estas máquinas son capaces de reconocer numerosos patrones de símbolos, los cuales identificaremos con la clase de los lenguajes regulares. Los conceptos que presentan los autómatas finitos y los lenguajes regulares son de interés fundamental para la mayoría de las aplicaciones que requieran técnicas de reconocimiento de patrones. Una de estas aplicaciones es la construcción de compiladores.

### 3.1. Autómatas Finitos Deterministas (*AFD*)

Esta máquina consiste en un dispositivo que puede estar en cualquiera de un número finito de estados, uno de los cuales es el estado inicial y por lo menos uno es estado de aceptación. A este dispositivo esta unido un flujo de entrada por medio del cual llegan en secuencia los símbolos de un alfabeto determinado. La máquina tiene la capacidad para detectar los símbolos conforme llegan y, basándose en el estado actual y el símbolo recibido, ejecutar una transición de estados, que consiste en un cambio a otro estado la permanencia en el estado actual.

La determinación de cual será precisamente la transición que ocurrirá al recibir un símbolo depende de un mecanismo de control de la máquina, programado para conocer cuál debe ser el nuevo estado dependiendo de la combinación del estado actual y el símbolo de entrada.

Tradicionalmente, un *AFD* se visualiza de la manera presentada en la figura 3.1

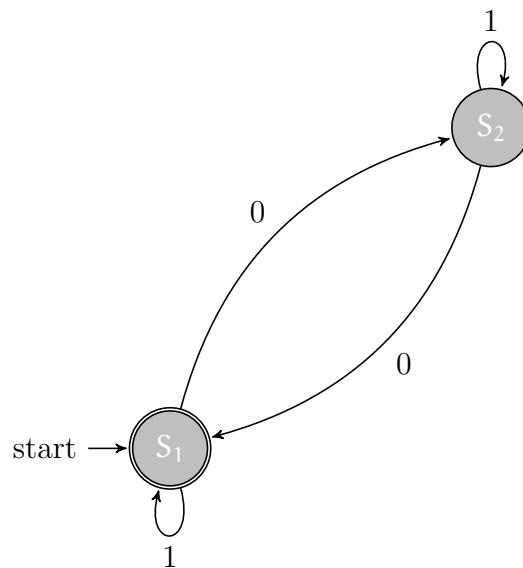


Figura 3.1: Autómata Finito Determinista

**Definición 3.1.** Un *AFD* es la quintupla:

$$M = (S, \Sigma, \delta, i, F)$$

donde:

- $S$  es un conjunto finito de estados.
  - $\Sigma$  es el alfabeto de la máquina.
  - $\delta : S \times \Sigma \rightarrow S$  es una función, llamada función de transición de estados.
  - $i$  (un elemento de  $S$ ) es el estado inicial.
  - $F$  (un subconjunto de  $S$ ) es el conjunto de estados de aceptación.
- 
- a. La función  $\delta$  describe el efecto que tiene esta entrada sobre los estados de la máquina. Así si la máquina se encuentra en el estado  $s_i$  y ocurre la entrada  $x$ , el siguiente estado de la máquina será  $\delta(s_i, x) = s_j$ .
  - b. El paso siguiente es extender la función  $\delta$  para que pueda describirse el estado en que termina el AFD si comienza en un estado  $s_i$  y recibe la cadena  $x_1, x_2, \dots, x_n$  de símbolos de entrada.

Así pues, la función

$$\delta^* : S \times \Sigma^* \rightarrow S$$

definida recursivamente por

$$\begin{aligned}\delta^*(s_i, x_1, x_2, \dots, x_n) &= \delta^*(\delta(s_i, x_1), x_2, \dots, x_n), \\ \delta^*(s_i, x_k) &= \delta(s_i, x_k)\end{aligned}$$

equivale exactamente a lo que esperábamos al obtener al evaluar una cadena de entradas  $x_1, x_2, \dots, x_n$  en  $\Sigma^*$ .

- c. Diremos que una cadena  $w \in \Sigma^*$  es aceptada o reconocida por el AFD  $M = (S, \Sigma, \delta, i, F)$  si y sólo si  $\delta^*(i, w) \in F$ .
- d. El conjunto de todas las cadenas aceptadas por la máquina  $M$  se denomina el lenguaje de la máquina lo que se denota con  $L(M)$ .

**Ejemplo 3.1.1.** Si definimos:

$$\begin{aligned} S &= \{s_0, s_1, s_2\} \\ \Sigma &= \{0, 1\} \\ F &= \{s_2\} \end{aligned}$$

además:

$$\begin{aligned} \delta(s_0, 0) &= s_0 \\ \delta(s_0, 1) &= s_1 \\ \delta(s_1, 0) &= s_1 \\ \delta(s_1, 1) &= s_2 \\ \delta(s_2, 0) &= s_2 \\ \delta(s_2, 1) &= s_0 \end{aligned}$$

se tiene que  $M = (S, \Sigma, \delta, i, F)$  es un AFD.

- La función de transiciones  $\delta$  puede representarse en un tabla que llamaremos tabla de transición de estados:
- Los AFD  $M = (S, \Sigma, \delta, i, F)$  definen en forma natural una relación  $R_M : S \rightarrow S$ , de acuerdo con :

$$s_i R_M s_j \Leftrightarrow \exists x (x \in \Sigma \wedge \delta(s_i, x) = s_j)$$

- El digrafo asociado a  $R_M$  le llamaremos diagrama de transiciones.
- figura...
- $01110 \in L(M)$  ?
- $L(M) = ?$

**Ejemplo 3.1.2.** Diseñar un AFD  $M$  tal que  $L(M) = \{a, ab, b, ba, aa\}$

**Ejemplo 3.1.3.** Diseñar un AFD  $M$  tal que  $L(M)$  conste de todas las cadenas que contengan los patrones  $abb$   $baa$ .

**Ejemplo 3.1.4.** Diseñar un AFD  $M$  tal que  $L(M) = L((11 \vee 110)^*0)$

**Definición 3.2.** Para cualquier alfabeto  $\Sigma$  existe un lenguaje que no es generado por un AFD  $M$ .

**Definición 3.3. Lema del bombeo**

Si el lenguaje de máquina de un AFD  $M$  contiene cadenas de la forma  $x^n y^n$  para  $n$  enteros positivos arbitrariamente grandes entonces debe contener cadenas de la forma  $x^r y^k$  donde  $r \neq k$ .

**Observación 3.1.1.** El teorema anterior determina que

$$L(M) = \{x^n y^n / n \geq 0\}$$

no se puede generar por un AFD  $M$ . Lo que muestra que el poder computacional de los AFD es limitado.

## 3.2. Autómatas Finitos no Deterministas (AFND)

Ejemplos como el 2.1.1-4 hacen pensar que encontrar un AFD que corresponda a una expresión regular dada puede ser tedioso, sin embargo si relajamos la condición del determinismo, el proceso de diseñar suele ser mucho más sencillo. Estos autómatas son los que pasamos a definir a continuación.

**Definición 3.4.** Un AFND es la quintupla

$$M = (S, \Sigma, \rho, i, F)$$

donde:

- $S$  es un conjunto finito de estados.

- $\Sigma$  es el alfabeto de la máquina.
- $\rho : S \times \Sigma \rightarrow S$  es una relación, llamada relación de transición de estados.
- $i$  (un elemento de  $S$ ) es el estado inicial.
- $F$  (un subconjunto de  $S$ ) es el conjunto de estados de aceptación.

**Ejemplo 3.2.1.** Diseñar un AFND  $M$  que acepte las cadenas en el alfabeto  $\Sigma = \{x, y\}$  que no contengan tres  $x$  consecutivas.

**Ejemplo 3.2.2.** Diseñar un AFND  $M$  tal que su lenguaje de máquina consista en las cadenas del alfabeto  $\Sigma = \{x, y, z\}$  donde el patrón  $xy$  siempre esté seguido por una  $w$  y al patrón  $yx$  siempre siga una  $z$ .

**Ejemplo 3.2.3.** Diseñar un AFND  $M$  tal que  $L(M) = L((ab \vee ba)^*aba)$

**Definición 3.5.** Para cada AFND  $M$  existe un AFD  $\tilde{M}$  tal que:

$$L(M) = L(\tilde{M})$$

**Observación 3.2.1.** A partir del teorema anterior muchos autores no distinguen entre AFND y AFD cuando analizan la aceptación de lenguajes. Por el contrario, únicamente hacen referencia a Autómatas Finitos (AF); con frecuencia haremos lo mismo.

**Ejemplo 3.2.4.** Dado el AFND  $M$  hallar un AFD  $\tilde{M}$  tal que

$$L(M) = L(\tilde{M})$$

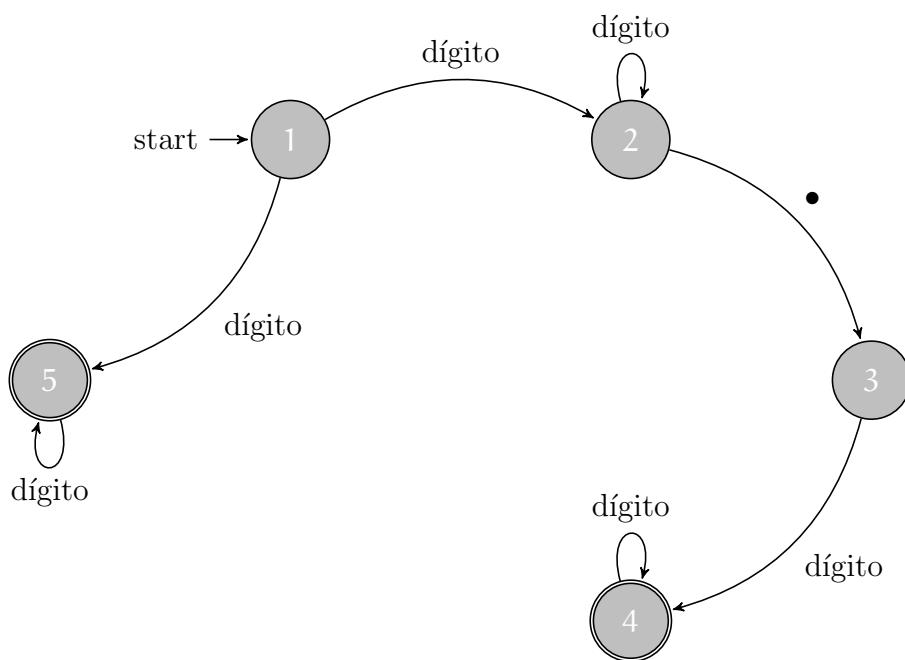


Figura 3.2: Diagrama de transición que acepta cadenas que representan enteros o cadenas que representan números reales en notación decimal

$$M = (S, \Sigma, \delta, i, F)$$

$$\tilde{M} = (P(S), \Sigma, \tilde{\delta}, \{i\}, \tilde{F})$$

**Solución:**

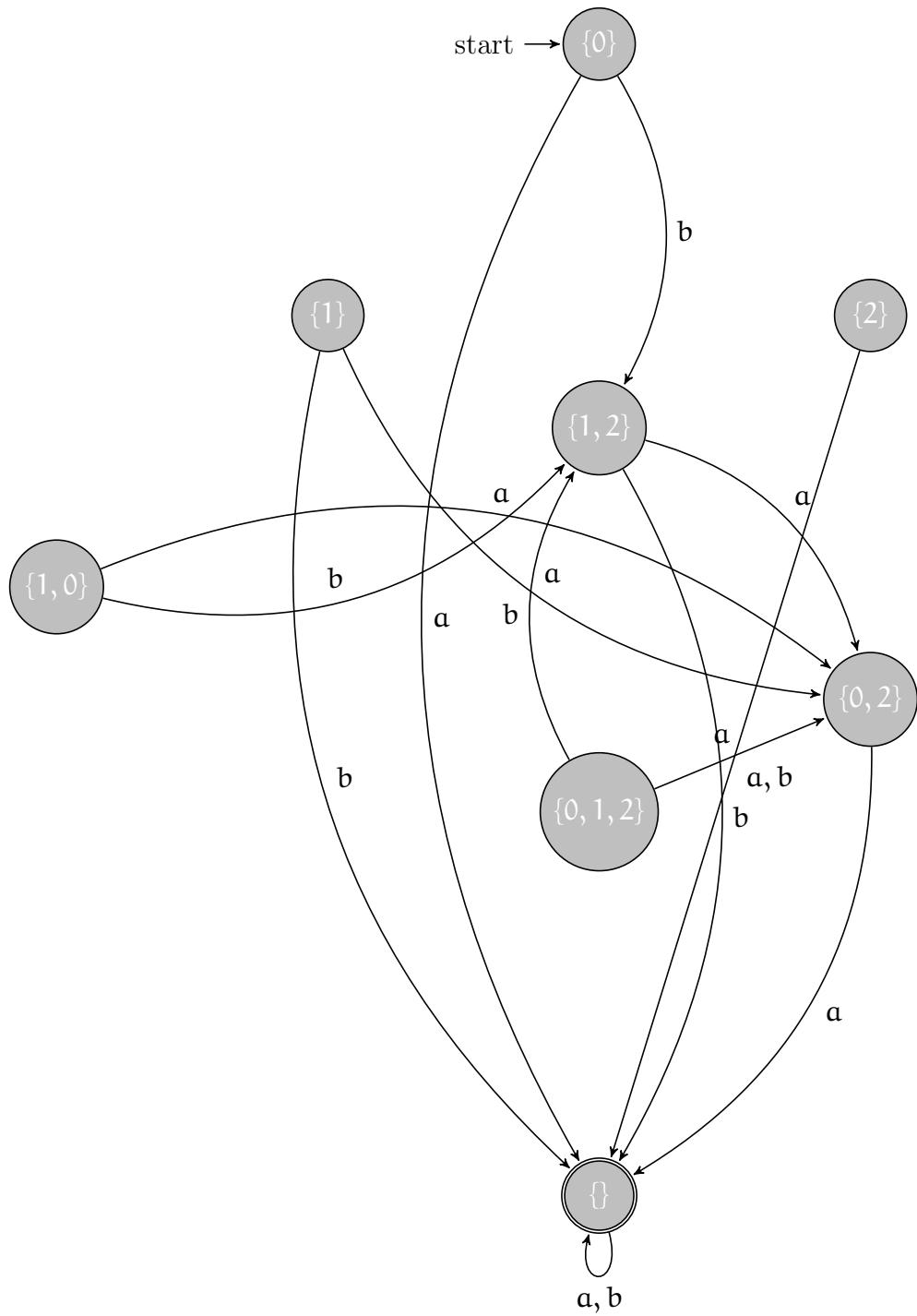


Figura 3.3: Diagrama de transición

**Teorema 3.1.** Los lenguajes regulares es decir los lenguajes generados por gramáticas del tipo 3 pueden ser generados por AF (expresiones regulares) y recíprocamente.

**Teorema 3.2.** La unión de dos lenguajes regulares es también un lenguaje regular

**Teorema 3.3.** La concatenación de dos lenguajes regulares es también un lenguaje regular.

**Teorema 3.4.** La estrella de Kleene de cualquier lenguaje regular es también un lenguaje regular.

### 3.3. Otras Formas del Lema del Bombeo

**Teorema 3.5.** Sea  $L$  un lenguaje regular recocido por un AF con  $n$  estados. Para cada  $x \in L$  con  $|x| \geq n$ ,  $x$  puede escribirse como  $x = uvw$  para algunas subcadenas  $u, v, w$  que satisfagan:  $|uv| \leq n$ ,  $|v| > 0$ , para cada  $m \geq 0$ ,  $uv^m w \in L$ .

**Teorema 3.6.** Sea  $L$  un lenguaje regular. Entonces existe un entero  $n$  tal que para cualquier  $x \in L$  con  $|x| \geq n$ , existen subcadenas  $u, v, w$  que satisfagan que:  $x = uvw$ ,  $|uv| \leq n$ ,  $|v| > 0$ , para cada  $m \geq 0$ ,  $uv^m w \in L$ .

**Ejemplo 3.3.1.** Demuestre que el lenguaje

$$L = \{0^i x / i \geq 0, x \in \{0, 1\}^*, |x| \leq i\}$$

no es regular.

**Solución:** Es frecuente que sea posible arreglárselas con una forma debilitada del lema de bombeo. Aquí se presentan dos versiones que omiten muchas de las conclusiones del teorema 3.5, pero todavía poseen fuerza suficiente para demostrar que ciertos lenguajes no son regulares.

**Teorema 3.7.** Sea  $L$  un lenguaje regular con infinitas cadenas. Entonces existe subcadenas  $u, v, w$  tales que:  $|v| > 0$ ,  $uv^m w \in L$ , para todo  $m \geq 0$ .

**Teorema 3.8.** *Sea  $L$  un lenguaje regular con infinitas cadenas. Entonces existen enteros  $p, q$  con  $q > 0$ , tales que para todo  $m \geq 0$ ,  $L$  contiene una cadena de longitud  $p + mq$ . En otras palabras, el conjunto de enteros: longitudes( $L$ ) =  $\{|x| / x \in L\}$  contiene la progresión aritmética de todos los enteros  $p + mq$ , para todo  $m \geq 0$ .*

**Ejemplo 3.3.2.** Demostrar que  $L = \{x^n / n \text{ es primo}\}$

### 3.4. Equivalencia y Minimización de Autómatas Finitos

**Definición 3.6.** Sea  $M = (S, \Sigma, \delta, i, F)$  un AF y sea  $R$  una relación de equivalencia en  $S$ . Se dice que  $R$  es una congruencia de máquinas en  $M$  si sólo si  $sRt \Leftrightarrow \delta(s, x)R\delta(t, x), \forall x : x \in \Sigma$ .

**Observación 3.4.1.** La relación  $R$  partitiona al conjunto de estados  $S$  en bloques de estados equivalentes ,estados que hacen la misma, tarea. Este hecho nos permite definir el AF  $\bar{M} = (\bar{S}, \Sigma, \bar{\delta}, [i], \bar{F})$  donde :  $\bar{S} = S|R = \{[s] / s \in S\}$ ,  $\bar{\delta} : \bar{S} \times \Sigma \rightarrow \bar{S}$  definida por  $\bar{\delta}([s], x) = [\delta(s, x)]$  es una función si  $R$  es una congruencia de máquinas en  $M$  y  $\bar{F} = \{f / f \in F\}$ .

Es claro que los AF  $M$  y  $\bar{M}$  sería equivalentes en el sentido de la relación de equivalencia  $R$ .

Por lo dicho debemos buscar una congruencia de máquinas  $R$  que haga que los AF  $M$  y  $\bar{M}$  sean equivalentes en el sentido de que  $L(M) = L(\bar{M})$ .

**Definición 3.7.** Sea  $M = (S, \Sigma, \delta, i, F)$  un AF y sea  $w \in \Sigma^*$ . Se dice que todo par de estados  $s, t$  de  $S$  son compatibles en  $w$  si y sólo si  $\delta(s, w) \cap \delta(t, w)$  ambos pertenecen a  $F$  ?,  $\bar{F}$ .

**Teorema 3.9.** *Sea  $M = (S, \Sigma, \delta, i, F)$  un AF. La relación de compatibilidad  $R$  en  $S$  definida por  $sRt \Leftrightarrow s, t$  son compatibles en  $w, \forall w : w \in \Sigma^*$  es una relación de congruencia de máquinas.*

**Teorema 3.10.** *Sea  $M = (S, \Sigma, \delta, i, F)$  un AF . Si  $R$  es una relación de compatibilidad en  $S$  entonces el AF  $\bar{M} = (\bar{S}, \Sigma, \bar{\delta}, [i], \bar{F})$  es tal que  $L(M) = L(\bar{M})$ .*

**Teorema 3.11.** Sea  $M = (S, \Sigma, \delta, i, F)$  un AF. La relación  $R_k$  en  $S$  definida por  $sR_k t \Leftrightarrow s, t$  son compatibles en  $w$ ,  $\forall w : w \in \Sigma^*$  y  $|w| \leq k$ , verifica:

$$R_{k+1} \sqsubseteq R_k, \quad k \geq 0$$

Toda  $R_k$  es una relación de equivalencia.

$R \sqsubseteq R_k$ , para toda  $k$ , donde  $R$  es la relación de compatibilidad.  $S|R_0 = \{F, \bar{F}\}$ .

$$sR_{k+1}t \Leftrightarrow sR_kt \text{ y } \delta(s, x) \text{ y } \delta(t, x), \quad \forall x : x \in \Sigma.$$

Si  $R_k = R_{k+1}$  para algún  $k > 0$  entonces  $R = R_k$

### 3.5. Lista de Ejercicios

1. Encuentre una expresión regular que corresponda a cada uno de los subconjuntos de  $\{0, 1\}^*$  que siguen :
  - a) El lenguaje de todas las cadenas que no contienen la subcadena 00 .
  - b) El lenguaje de todas las cadenas que contienen la cadena 00 no más de una vez .
2. En el lenguaje de programación C , todas las expresiones siguientes son válidas : 3 , 13. , .234 , 43.45 , +23.15 , +0 , -01 , -12.3 , 2e12 , +1.3e6 , -2.e+5 , 01E-06 , -.4E-7 , 00e0 . Con base en estos ejemplos , escriba una gramática de estructura de frases G que genere el lenguaje de todas las expresiones .
3. Construir un AFD mínimo que reconozca las palabras sobre  $\Sigma = \{a, b, c, d\}$  que contienen un número par de apariciones de la subcadena bcd .

4. Decida si los siguientes lenguajes son regulares o no . Demuestre que su respuesta es correcta.
- El conjunto de las cadenas de longitud impar sobre  $\Sigma = \{0, 1\}$  cuyo símbolo central es el 0 .
  - $L = \{a^{2^n} : n \geq 1\}$
5. Si  $L$  es un lenguaje regular sobre el alfabeto  $\Sigma = \{x, y\}$ . Demuestre que  $L_1 = \{w : w \in \Sigma^*, wx \in L\}$  es también un lenguaje regular .
6. Demostrar o dar un contraejemplo de las siguientes afirmaciones. Se supone que todos los lenguajes a los que se hace referencia son sobre el mismo alfabeto.
- La intersección de  $n$  conjuntos regulares es regular.
  - Todo conjunto finito es regular.
  - La unión infinita de conjuntos no regulares no puede ser regular.
  - La unión de conjuntos no regulares no puede ser regular.
  - Todo conjunto cuyo complementario sea finito es regular.
7. Dado el AFD , obtener el autómata mínimo.

	0	1
$\rightarrow s_0$	$s_1$	$s_5$
$s_1$	$s_6$	$s_2$
$*s_2$	$s_0$	$s_2$
$s_3$	$s_2$	$s_6$
$s_4$	$s_7$	$s_5$
$s_5$	$s_2$	$s_6$
$s_6$	$s_6$	$s_4$
$s_7$	$s_6$	$s_2$

el estado señalado con {\*} es de aceptación.

8. Demostrar que el lenguaje de todas las cadenas formadas por ceros y unos tales que cada pareja de ceros adyacentes aparece antes que cualquier pareja de unos adyacentes, es regular .
9. Obtener una expresión regular para representar los salarios que pueden aparecer en los anuncios de trabajo. Considere que los salarios pueden especificarse por hora, semana, mes o año. Pueden aparecer con

el símbolo de euro, o de cualquier otra unidad monetaria, como por ejemplo , dólar. Pueden incluir una o varias palabras que identifiquen que se trata de un salario. Sugerencia : mire las páginas de anuncios de un periódico o las ofertas de trabajo en la web para hacerse una idea de cuáles serán los patrones más útiles.

10. Si  $L$  es un lenguaje y  $a$  es un símbolo.  $L/a$  es el lenguaje que consta de las cadenas  $w$  tales que  $wa$  pertenece a  $L$  .  $a/L$  es el lenguaje que consta de las cadenas tales que  $aw$  pertenece a  $L$ .
    - a) Demostrar que si  $L$  es regular también lo es  $a/L$  .
    - b)  $\cup(L/a)\{a\} = L$  ? .
    - c)  $a/L$  se llama la derivada y se escribe  $\frac{dL}{da}$  . Estas derivadas se aplican a las expresiones regulares de forma similar a como se aplica la derivada común. Así, si  $R$  es una expresión regular representara lo mismo que  $\frac{dL}{da}$  si  $L = L(R)$ , Demostrar  $\frac{d(R+S)}{da} = \frac{dR}{da} + \frac{dS}{da}$
  11. Muestre que el lenguaje  $L = \{ww / w \in \{0, 1\}^*\}$  no es regular.
  12. Construya los AF que acepten los lenguajes del alfabeto  $\Sigma = \{0, 1\}$ 
    - a) El conjunto de todas las cadenas en las que cada bloque de cinco símbolos consecutivos contiene al menos dos ceros.
    - b) El conjunto de todas las cadenas que comienzan por 1 y que, si se interpretan como la representación binaria de un entero, sean múltiplos de 5. Por ejemplo, las cadenas 101, 1010 y 1111 pertenecen al lenguaje; 0, 100 y 111 no pertenecen.
  13. Demostrar que el lenguaje no es regular
- $$L = \{a^n / n = 2^k, \text{ para algún entero } k \geq 0\}$$
14. Si  $x$  es una palabra cualquiera, se denota por  $ss(x)$  al conjunto de cadenas obtenido mediante la eliminación de un número arbitrario, incluido el cero, de símbolos de dicha palabra  $x$ . Asimismo, si  $L$  es un lenguaje,  $ss(L) = \{ss(x) / x \in L\}$  :
    - a) Calcular :  $ss(\{x / x \in a^+b^+, |x| \leq 3\})$  .
    - b) Calcular :  $ss(a^+b^+)$  .
    - c) Demostrar que si  $L$  es un lenguaje regular, entonces  $ss(L)$  es un lenguaje regular.

15. Para los lenguajes dados sobre  $\Sigma = \{a, b\}$  construir una expresión regular de él y un Autómata Finito que lo acepte:
16. Cómo puede alterarse el AFD  $M = (S, \Sigma, \delta, i, F)$  para obtener un AFD que acepte el lenguaje  $\Sigma^* - L(M)$ ?
17. Demuestre que si  $L_1$  y  $L_2$  son lenguajes regulares, entonces  $L_1 \cap L_2$  es regular.
18. Demuestre que si  $L$  es regular, entonces también es regular el lenguaje  $L^{-1}$ .
19. Demuestre que  $L = \{x^n / n \text{ es primo}\}$  no es un lenguaje regular.
20. Sea  $\Sigma = \{a, b\}$ 
  - a) Construir AFD que acepten  $L(a^*b)$  y  $L(ab^*)$ .
  - b) A partir de los AFD de la parte (a), construir los AFD que acepten  $\Sigma^* - L(a^*b)$  y  $\Sigma^* - L(ab^*)$ .
  - c) A partir de los AFD de la parte (b), construir los AFD que acepten  $(\Sigma^* - L(a^*b)) \cup (\Sigma^* - L(ab^*))$ .
  - d) Usar el resultado de la parte (c) para construir un AFD que acepte el lenguaje  $L(a^*b) \cap L(ab^*)$ .

# Capítulo 4

## Autómatas de Pila y Lenguajes Independientes del Contexto

Así como los lenguajes regulares tienen un autómata equivalente (autómata finito) las gramáticas independientes del contexto tienen su contraparte en forma de máquina, es decir, el autómata de pila. En este caso la equivalencia es un poco menos satisfactoria, ya que el autómata de pila es un dispositivo no determinístico y su versión determinística solo acepta un subconjunto de todos los lenguajes independientes del contexto. Afortunadamente, este subconjunto incluye a la sintaxis de la mayoría de los lenguajes de programación.

Un autómata de pila es esencialmente un autómata finito al que agregamos un sistema de memoria interna (en forma de pila).

**Definición 4.1.** Un autómata de pila es una sextupla de la forma

$$M = (S, \Sigma, \Gamma, \rho, s_0, F),$$

donde

- $S$  es una colección finita de estados

- $\Sigma$  es una alfabeto llamado alfabeto de entrada
- $\Gamma$  es un alfabeto, conocido como alfabeto de la pila
- $\rho$  es una relación de transición de  $S \times (\Sigma \cup \Gamma)$  en  $S \times \Gamma$

**Observación 4.0.1.** Notemos lo siguiente

- a) La principal diferencia entre los autómatas de pila y los finitos es que los primeros cuentan con una pila en donde pueden almacenar información para recuperarla más tarde.

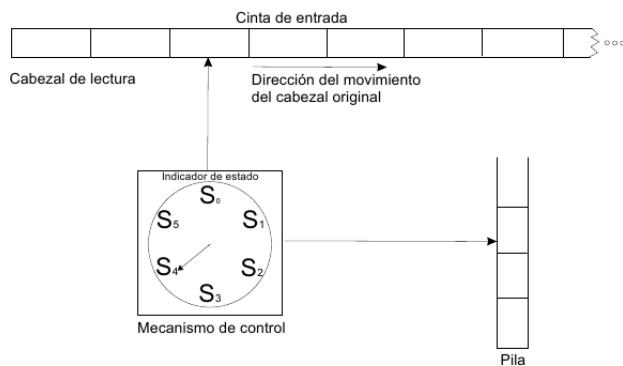


Figura 4.1:

- b) El alfabeto de la pila puede incluir algunos o todos los símbolos del alfabeto de entrada y quizás algunos símbolos adicionales que la máquina utiliza como marcas internas. Una máquina podrá almacenar símbolos especiales en su pila para separar secciones que tengan interpretaciones distintas. Esto es, si un máquina inserta un símbolo especial en la pila antes de efectuarse algún otro cálculo, entonces la presencia de este símbolo en la cima de la pila puede usarse como indicador de “pila vacía” para cálculos
- c) Las transiciones  $\rho$  que ejecutan F los autómatas de pila deben ser variantes de la siguiente secuencia básica:
1. Leer un símbolo de la entrada
  2. Extraer un símbolo del apila
  3. Insertar un símbolo en la pila y pasar a un nuevo estado.

Este proceso se representa con la notación:

$$(p, x, s; q, y)$$

que indica, el estado actual y el símbolo del alfabeto de entrada que se lee, el símbolo que se extrae de la pila; el nuevo estado y el símbolo que se inserta en la pila

**Ejemplo 4.0.1.** Los siguientes ejemplos tratan sobre autómatas de pila:

- a)  $(p, \lambda, \lambda; q, \lambda)$  significa que pasa del estado  $p$  al estado  $q$ ; al encontrarse en el estado  $p$ , la máquina podría no avanzar su cabezal de lectura (lo que consideramos como la lectura de la cadena vacía), no extraer un símbolo de su pila (extraer la cadena vacía), no insertar un símbolo en su pila (insertar cadena vacía) y pasar al estado  $q$ .
- b)  $(p, \lambda, s; q, \lambda)$  significa que pasa del estado  $p$  al estado  $q$  extrayendo el símbolos de la pila.
- c)  $(p, x, \lambda; q, z)$  significa que pasa del estado  $p$  al estado  $q$  leyendo la entrada  $x$ , sin extraer ningún símbolo de su pila e insertando en la pila el símbolo  $z$ .
- d)  $(p, \lambda, \lambda; q, z)$  significa que pasa del estado  $p$  al estado  $q$  e inserta en su pila el símbolo  $z$ .

**Observación 4.0.2.** En los autómatas de pila

- a) Las transiciones  $(p, x, s; q, y)$  se representan por:

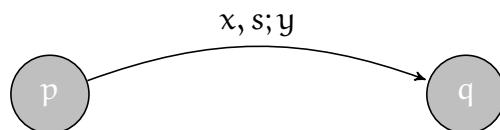


Figura 4.2: Transiciones  $(p, x, s; q, y)$

- b) Para analizar una cadena, ésta se coloca en la cinta de entrada; el cabezal de lectura se coloca al extremo izquierdo de la cinta, luego se

pone en marcha la máquina desde su estado inicial, con la pila vacía, y declaramos que la cadena es aceptada si es posible (pues nuestro autómata es no determinista) que la máquina llegue a un estado de aceptación después de leer toda la cadena. Esto no quiere decir que la máquina deba encontrarse en un estado de aceptación inmediatamente después de leer el último símbolo de la cadena de entrada, sino que significa que ya no tiene que leer más en la cinta. Por ejemplo, después de leer el último símbolo de la entrada, un autómata de pila puede ejecutar varias transiciones de la forma  $(p, \lambda, x; q, y)$  antes de aceptar la cadena.

- c) Si se restringe las transiciones disponibles para un autómata de pila a la forma  $(p, x, \lambda; q, y)$ , tenemos la clase de los autómatas finitos. Por lo tanto, los lenguajes aceptados por los autómatas de pila incluyen los lenguajes regulares.

**Ejemplo 4.0.2.** Dado el autómata de pila

$$M = (S, \Sigma, \Gamma, \rho, s_0, F)$$

representado por el diagrama de transiciones:

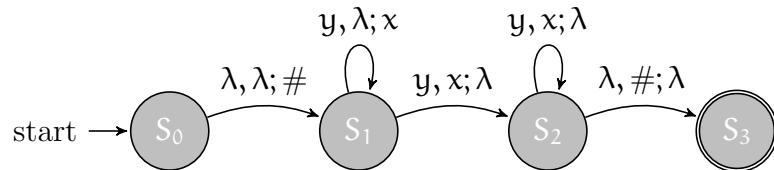


Figura 4.3: Diagrama de transición para  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$

- a) ¿La cadena  $xxxxyyyy$  pertenece a  $L(M)$ ?
- b) Describa el lenguaje  $L(M)$

**Solución:**

- a) Diagrama de estados

Contenido de la pila B	Resto de la entrada	Transición terminal
$\lambda$	$xxxxyyyy$	$(s_0, \lambda, \lambda; s_1, \#)$
#	$xxxxyyyy$	$(s_0, x, \lambda; s_1, x)$
$x\#$	$xxxyyy$	$(s_1, x, \lambda; s_1, x)$
$xx\#$	$xxyyyy$	$(s_1, x, \lambda; s_1, x)$
$xxx\#$	$xyyyy$	$(s_1, x, \lambda; s_1, x)$
$xxxx\#$	$y yyyy$	$(s_1, y, x; s_2, \lambda)$
$xxx\#$	$yyy$	$(s_2, y, x; s_2, \lambda)$
$xx\#$	$yy$	$(s_2, y, x; s_2, \lambda)$
$x\#$	$y$	$(s_2, y, x; s_2, \lambda)$
#	$\lambda$	$(s_2, \lambda, \#; s_3, \lambda)$

Por lo tanto,  $xxxxyyyy \in L(M)$

- b) De que se deduce que el número de  $x$  debe ser igual al número de  $y$  y como el estado inicial  $s_0$  es de aceptación  $\lambda \in L(M)$ , es decir:

$$L(M) = \{x^n y^n / n \in \mathbb{N}\}$$

**Ejemplo 4.0.3.** Dado el autómata de pila

$$M = (S, \Sigma, \Gamma, \rho, s_0, F)$$

representado por el diagrama de transiciones

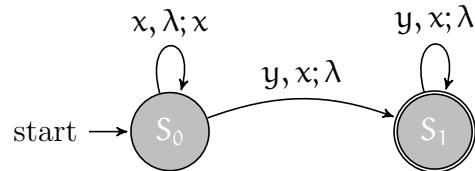


Figura 4.4: Diagrama de transición para  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$

- a) ¿La cadena  $xxxxyy$  pertenece a  $L(M)$ ?  
 b) Describa el lenguaje  $L(M)$

**Solución:**

a) Diagrama de estados

Contenido de la pila B	Resto de la entrada	Transición terminal
$\lambda$	$xxxxyy$	$(s_0, x, \lambda; s_0, x)$
$x\#$	$xxyy$	$(s_0, x, \lambda; s_0, x)$
$xx\#$	$xyy$	$(s_0, x, \lambda; s_0, x)$
$xxx\#$	$yy$	$(s_0, y, x; s_1, \lambda)$
$xx\#$	$y$	$(s_1, y, x; s_1, \lambda)$
$x\#$	$\lambda$	

Por lo tanto,  $xxxxyy \in L(M)$

b) Un análisis similar nos permitirá deducir por ejemplo que la cadena  $xxyy$  no pertenece a  $L(M)$ . De donde:

$$L(M) = \{x^n y^m / n, m \in \mathbb{Z}^+, m \geq n\}$$

**Observación 4.0.3.** Observe que

- a) En el ejemplo 4.0.2 la máquina acepta una cadena vaciando totalmente su pila y que en el ejemplo 4.0.3 la máquina acepta cadenas sin vaciar su pila.
- b) Se podría conjeturar que la aplicación de una teoría basada en estos autómatas nos llevaría a módulos de programa que devolvieran el control a otros módulos dejando en la pila residuos de sus cálculos que podría ocasionar confusiones en cálculos posteriores.

**Ejemplo 4.0.4.** El siguiente autómata de pila acepta el mismo lenguaje que el autómata de pila del ejemplo 4.0.3, pero vacía su pila antes de llegar a un estado de aceptación

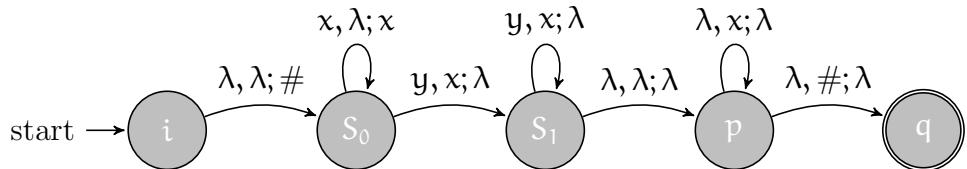


Figura 4.5: Diagrama de transición para  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$

**Teorema 4.1.** *Para cada autómata de pila que acepta cadenas sin vaciar su pila, existe un autómata que acepta el mismo lenguaje pero que vacía su pila antes de llegar a un estado de aceptación.*

*Demostración.* Si  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$  es un autómata de pila que acepta cadenas sin tener que vaciar necesariamente su pila, entonces si modificamos  $M$  de la siguiente manera:

- Añada un nuevo estado, designándolo como inicial, y una transición que permita a la nueva máquina  $M'$  pasar del nuevo estado inicial al anterior de  $M$  a la vez que inserta en la pila un símbolo especial  $\#$  (que no se encontraba anteriormente en  $\Gamma$ ).
- Elimine la característica de aceptación de cada estado de aceptación de  $M$ . Luego añada un estado  $p$  junto con las transiciones que permiten a la máquina pasar de cada uno de los antiguos estados de aceptación a  $p$  sin leer, extraer o insertar un símbolo.
- Para cada  $x \in \Sigma$  (sin incluir  $\#$ ), introduzca la transición  $(p, \lambda, x; p, \lambda)$ .
- Añada un nuevo estado de aceptación  $q$  y la transición  $(p, \lambda, \#; q, \lambda)$ .

Luego es claro que:  $M' = (S', \Sigma', \Gamma', \rho', s'_0, F')$  es tal que  $L(M') = L(M)$ , sólo que la máquina  $M'$  llega a su estado de aceptación únicamente cuando su pila está vacía.  $\square$

**Observación 4.0.4.** Observe que

- a) La gramática independiente del contexto

$$G = (N, \Sigma, v_0, \rightarrow)$$

donde:

$$\begin{aligned} V &= \{v_0, x, y\} \\ \Sigma &= \{x, y\} \end{aligned}$$

$$\begin{array}{lcl} \mapsto & : & v_0 \mapsto xv_0y \\ & & v_0 \mapsto \lambda \end{array}$$

que evidentemente no es regular, genera el lenguaje

$$L(G) = \{x^n y^m / n \in \mathbb{N}\}$$

Este ejemplo y el ejemplo 2, cambiando con el hecho de que cualquier gramática regular es una gramática independiente del contexto, permite demostrar que los autómatas de pila generan una mayor colección de lenguajes que los autómatas finitos.

- b) La gramática de inserción múltiple  $(p, a, s; q, xyz)$  significa que se insertan en la pila los símbolos  $z, y, x$ , es decir

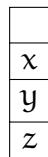


Figura 4.6: Pila con elementos  $x, y, z$

La transición de inserción múltiple  $(p, a, s; q, xyz)$  podría simularse con la secuencia de transiciones  $(p, a, s; q - 1, z)$ ,  $(q_1, \lambda, \lambda; q_2, y)$  y  $(q_2, \lambda, \lambda; q, x)$  donde  $q_1$  y  $q_2$  son estados adicionales a los que no pueden llegar ninguna otra secuencia de transiciones, es decir, que las transiciones de inserción múltiple no añaden capacidades adicionales a la máquina.

**Ejemplo 4.0.5.** Diseñe un autómata de pila  $M$  tal que:

$$L(M) = \{x^n y^{2n} / n \in \mathbb{Z}^+\}$$

**Solución:**

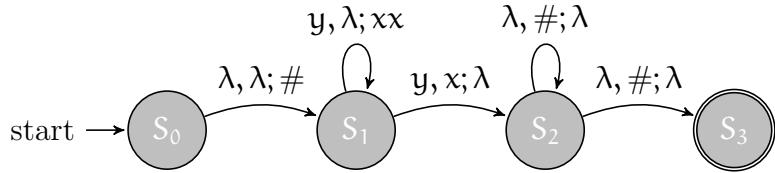


Figura 4.7: Autómata de pila

**Teorema 4.2.** *Para cada gramática G independiente del contexto, existe un autómata de pila M tal que L(G) = L(M).*

*Demostración.* Sea  $G = (N, \Sigma, v_0, \rightarrow)$  una gramática independiente del contexto donde  $V$  es la unión de los símbolos terminales,  $\Sigma$  y los no terminales  $N$ . Entonces, se construye un autómata de pila  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$  de la manera siguiente:

- $\Sigma, \Gamma = V \cup \{\#\}$ ,  $S = \{s_0, p, q, f\}$ ,  $F = \{f\}$ .
- $(s_0, \lambda, \lambda; p, \#)$
- $(p, \lambda, \lambda; q, v_0)$
- $(q, \lambda, v; q, w) \Leftrightarrow v \mapsto w, w \in V^*$
- $(q, x, x; q, \lambda) \Leftrightarrow x \in \Sigma$
- $(q, \lambda, \#; f, \lambda)$

□

Un autómata de pila construido de esta manera analizará una cadena de entrada marcando primero el fondo de la pila con el símbolo  $\#$ , luego insertando en la pila el símbolo inicial de la gramática y después entrando al estado  $q$ . De ahí y hasta que el símbolo  $\#$  vuelve a aparecer en la cima de la pila, el autómata extraerá un no terminal de la pila y lo reemplazará con el lado derecho de una regla de producción aplicable o extraerá un terminal de la pila a la vez que lee el mismo terminal en la entrada. Una vez que el símbolo  $\#$  regresa a la cima de la pila el autómata cambiará a su estado de aceptación  $f$ , indicando que la entrada recibida hasta ese punto es aceptable.

Observe que la cadena de símbolos que integran la parte derecha de una regla de producción se inserta en la pila de derecha a izquierda. Así el no

terminal situado más, a la izquierda será el primero en surgir en la cima de la pila; por lo tanto, también será el primer no terminal de la pila que se reemplazará. Por consiguiente, el autómata analiza su entrada efectuando una derivación por la izquierda de acuerdo con las reglas de la gramática en la cual se basa. Entonces, el autómata acepta exactamente el mismo lenguaje que genera la gramática, es decir, que  $L(M) = L(G)$ .

**Ejemplo 4.0.6.** Dada la gramática independiente del contexto

$$G = (N, \Sigma, v_0, \rightarrow)$$

donde:

$$\begin{aligned} < v_0 > ::= z < v_1 > < v_2 > z \\ < v_1 > ::= a < v_1 > a | z \\ < v_2 > ::= b < v_2 > b | z \end{aligned}$$

Hallar un autómata de pila  $M$  tal que

$$L(G) = L(M)$$

**Solución:**

Primero, observe que

$$L(G) = \{za^nza^nb^mzb^mz / n, m \in \mathbb{N}\}$$

Luego

$$M = (S, \Sigma, \Gamma, \rho, s_0, F)$$

se define por:

$$\Sigma = \{a, b, z\}, \Gamma = \{v_0, v_1, v_2, a, b, z, \#\}, S = \{s_0, p, q, f\}, F = \{f\}$$

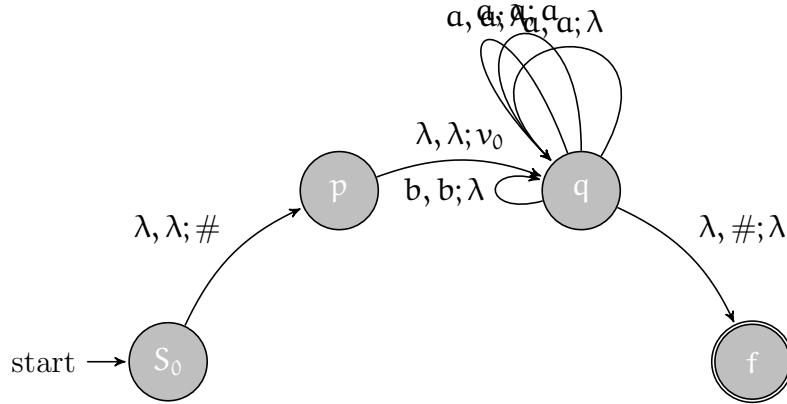


Figura 4.8: Autómata de pila

**Teorema 4.3.** Para cada autómata de pila  $M$ , existe una gramática  $G$  independiente del contexto tal que  $L(M) = L(G)$ .

*Demostración.* Representamos con  $\langle p, x, q \rangle$  el hecho de pasar del estado  $p$  al estado  $q$  de manera que el símbolo  $x$  se elimina de la parte superior de la pila. Ahora, sin perdida de generalidad podemos suponer que el autómata de pila  $M = (S, \Sigma, \Gamma, \rho, s_0, F)$  acepta cadenas únicamente cuando su pila está vacía. Entonces se construye una gramática

$$G = (N, \Sigma, v_0, \rightarrow)$$

de la siguiente manera:

$\Sigma$  = Símbolos terminales

$N$  = Símbolos no terminales

$V = N \cup T$

$\rightarrow$ :

- Para cada estado de aceptación  $f \in F$ , se defina:

$$v_0 \mapsto \langle s_0, \lambda, f \rangle$$

(cualquier derivación de esta gramática comenzará sustituyendo el símbolo inicial de la gramática por un objetivo principal del autómata).

- Para cada estado  $p \in S$ , defina:

$$\langle p, \lambda, p \rangle \mapsto \lambda$$

(puede eliminarse el objetivo de pasar de un estado a sí mismo sin modificar la pila).

- Para cada transición  $(p, x, y; q, z)$  con  $y \neq \lambda$ , defina:

$$\langle p, y, r \rangle \mapsto x \langle q, z, r \rangle, \forall r \in S$$

(indican que el objetivo de pasar de un estado  $p$  a un estado  $x$  eliminando  $y$  y de la pila puede lograrse si se pasa primero a un estado  $q$  mientras que se lee  $x$  de la entrada y se intercambia  $z$  por  $y$  en la pila).

- Para cada transición  $(p, x, \lambda; q, z)$ , defina:

$$\langle p, w, r \rangle \mapsto x \langle q, z, k \rangle | k, w, r \rangle, w \in (\Gamma \cup \{\lambda\}), k, r \in S$$

(reflejan que el objetivo de pasar de un estado  $p$  a un estado  $r$  eliminando  $w$  de la pila puede lograrse si primero el pasa al estado  $q$  mientras se lee  $x$  de la entrada y si se inserta  $z$  en la pila, por medio de la transición  $(p, x\lambda; q, z)$ , y luego se intenta pasar del estado  $q$  al estado  $r$  a través de un estado  $k$ , a la vez que se eliminan  $z$  y  $w$  de la pila).

Observe que las producciones que hemos definido forman una gramática independiente del contexto, que genera el mismo lenguaje que acepta el autómata.  $\square$

**Ejemplo 4.0.7.** Dado el autómata de pila:

- a) Determinar  $L(M)$

- b) Hallar una gramática independiente del contexto G, tal que  $L(G) = L(M)$

**Solución:**

a)  $L(M) = \{cb^n c / n \in \mathbb{N}\}$

- b) Se define la gramática  $G = (N, T, v_0, \rightarrow)$  de la siguiente manera:

$$T = \{c, b\}, N = \{f, g, h, v_0\}, V = N \cup T$$

$\rightarrow$ :

•

$$v_0 \rightarrow \langle f, \lambda, h \rangle$$

•

$$\langle f, \lambda, f \rangle \rightarrow \lambda$$

$$\langle g, \lambda, g \rangle \rightarrow \lambda$$

$$\langle h, \lambda, h \rangle \rightarrow \lambda$$

- Para la transición  $(g, c, c; h, \lambda)$

$$\langle g, c, f \rangle \rightarrow c \langle h, \lambda, f \rangle$$

$$\langle g, c, g \rangle \rightarrow c \langle h, \lambda, g \rangle$$

$$\langle g, c, h \rangle \rightarrow c \langle h, \lambda, h \rangle$$

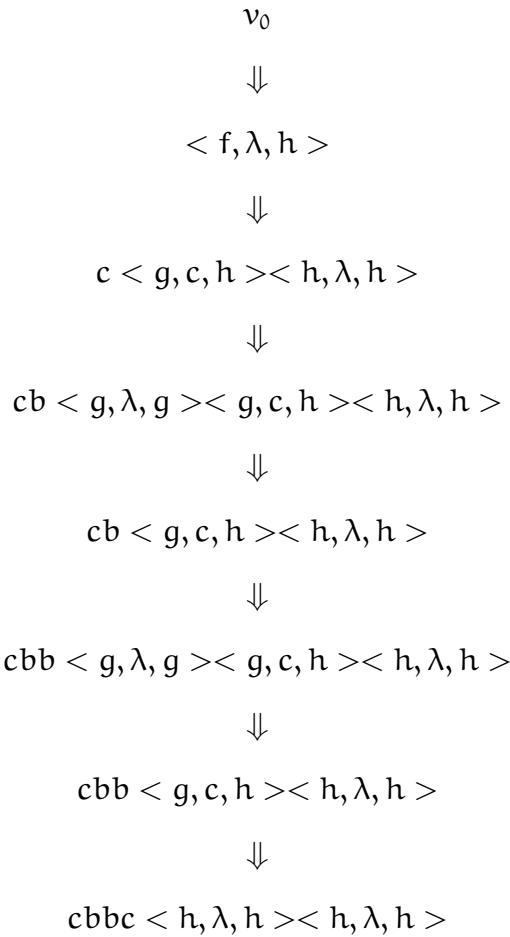
- Para la transición  $(f, c, \lambda; g, c)$

$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, f \rangle < f, \lambda, f \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, f \rangle < f, c, f \rangle$
$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, g \rangle < g, \lambda, f \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, g \rangle < g, c, f \rangle$
$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, h \rangle < h, \lambda, f \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, h \rangle < h, c, f \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, f \rangle < f, \lambda, g \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, f \rangle < f, c, g \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, g \rangle < g, \lambda, g \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, g \rangle < g, c, g \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, h \rangle < h, \lambda, g \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, h \rangle < h, c, g \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, f \rangle < f, \lambda, h \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, f \rangle < f, c, h \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, g \rangle < g, \lambda, h \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, g \rangle < g, c, h \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, h \rangle < h, \lambda, h \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, h \rangle < h, c, h \rangle$

- Para la transición  $(g, b, \lambda; g, \lambda)$

$< g, \lambda, f > \mapsto b < g, \lambda, f > < f, \lambda, f >$	$< g, c, f > \mapsto b < g, \lambda, f > < f, c, f >$
$< g, \lambda, f > \mapsto b < g, \lambda, g > < g, \lambda, f >$	$< g, c, f > \mapsto b < g, \lambda, g > < g, c, f >$
$< g, \lambda, f > \mapsto b < g, \lambda, h > < h, \lambda, f >$	$< g, c, f > \mapsto b < g, \lambda, h > < h, c, f >$
$< g, \lambda, g > \mapsto b < g, \lambda, f > < f, \lambda, g >$	$< g, c, g > \mapsto b < g, \lambda, f > < f, c, g >$
$< g, \lambda, g > \mapsto b < g, \lambda, g > < g, \lambda, g >$	$< g, c, g > \mapsto b < g, \lambda, g > < g, c, g >$
$< g, \lambda, g > \mapsto b < g, \lambda, h > < h, \lambda, g >$	$< g, c, g > \mapsto b < g, \lambda, h > < h, c, g >$
$< g, \lambda, h > \mapsto b < g, \lambda, f > < f, \lambda, h >$	$< g, c, h > \mapsto b < g, \lambda, f > < f, c, h >$
$< g, \lambda, h > \mapsto b < g, \lambda, g > < g, \lambda, h >$	$< g, c, h > \mapsto b < g, \lambda, g > < g, c, h >$
$< g, \lambda, h > \mapsto b < g, \lambda, h > < h, \lambda, h >$	$< g, c, h > \mapsto b < g, \lambda, h > < h, c, h >$

De acuerdo con el teorema 1.8, esta gramática, es tal que  $L(G) = L(M)$ . Observe, por ejemplo que  $cbbc \in L(M)$  se deriva con las producciones de esta gramática.



$$\begin{array}{c}
 \downarrow \\
 \text{cbbc} < h, \lambda, h > \\
 \downarrow \\
 \text{cbbc}
 \end{array}$$

## 4.1. Forma normal Chomsky

A primera vista parece que la flexibilidad que permiten las gramáticas independientes del contexto impone pocas restricciones a las posibles estructuras de cadenas que pueden encontrarse en los lenguajes independientes del contexto. Pero los lenguajes independientes del contexto si tienen gramáticas cuyas producciones se adhieren a formatos extremadamente rígidos, por lo que, se hace necesario investigar las producciones de las gramáticas independientes del contexto.

Si el lenguaje generado por la gramática contiene la cadena vacía, entonces debe aparecer cuando menos una regla  $v \mapsto \lambda$  en la gramática. Sin embargo, podríamos preguntar hasta qué punto se requieren las producciones  $\lambda(v \mapsto \lambda)$ , si  $G$  es una gramática independiente del contexto que no genera la cadena vacía, para ver esto:

Primero, démonos cuenta que la existencia de una producción  $\lambda$  puede permitir que más no terminales que el que aparece en la regla puedan reescribirse como la cadena vacía. Por ejemplo si una gramática contiene las reglas  $v_1 \mapsto \lambda$  y  $v_2 \mapsto v_1$ , entonces tanto  $v_1$  como  $v_2$  se podrían reescribirse como  $\lambda$ . Para identificar el efecto de las producciones  $\lambda$  en una gramática independiente del contexto, debemos aislar todos los no terminales que pueden reescribirse como la cadena vacía. Luego:

- Se define un encuadernamiento  $\lambda$  de longitud  $n$ , como la secuencia de producciones de la forma  $v_n \mapsto v_{n-1}, v_{n-1} \mapsto v_{n-2}, v_{n-2} \mapsto v_{n-3}, \dots, v_0 \mapsto \lambda$ ; el no terminal  $v_n$  se llama el origen del encuadernamiento.

- Se define,  $U_0$  como el conjunto de los no terminales que aparecen como origen de los encuadernamientos  $\lambda$  de longitud cero, es decir, los no terminales que aparecen del lado izquierdo de las producciones  $\lambda$ . A este conjunto le añadimos los orígenes de todos los encuadernamientos  $\lambda$  de longitud uno para formar el conjunto  $U_1$ . Luego, a  $U_1$  le agregamos los orígenes de todos los encuadernamientos  $\lambda$  de longitud dos para obtener el conjunto  $U_2$  y así sucesivamente;
- Puesto que en una gramática sólo hay un número finito de no terminales, debe existir un punto en el cual este proceso deje de introducir no terminales adicionales. Al llegar a este punto hemos recopilado todos los no terminales de  $G$  que pueden reescribirse como la cadena vacía, este conjunto se representa por  $U$ . (observe que , como  $G$  no genera la cadena vacía  $U$  no contiene el símbolo inicial  $v_0$  de  $G$ ).
- Para cada producción de la forma  $v \mapsto w$ , donde  $w$  es una cadena de terminales y no terminales, agregamos a  $G$  todas las producciones de la forma  $v \mapsto w'$ , donde  $w'$  es cualquier cadena no vacía obtenida al eliminar de  $w$  una o más ocurrencias de no terminales en  $U$ .

Una vez que agregamos estas producciones a la gramática, ya no necesitamos las producciones  $\lambda$  y no reducimos los poderes generativos de la gramática. Por lo tanto, concluimos que cualquier lenguaje independiente del contexto que no contiene la cadena vacía se puede generar por medio de una gramática independiente del contexto que no tenga producciones  $\lambda$ .

**Teorema 4.4.** *Si  $L$  es un lenguaje independiente del contexto que no contiene la cadena vacía, entonces existe una gramática  $G$  independiente del contexto tal que  $L(G) = L$  y el lado derecho de cualquier producción de  $G$  consiste en una sólo símbolo terminal o exactamente los símbolos no terminales.*

*Demostración.* Sea  $L$  un lenguaje independiente del contexto que no contiene la cadena vacía. Sin pérdida de generalidad podemos afirmar que existe una gramática independiente del contexto  $G$  que no contiene producciones  $\lambda$  que genera  $L$ .

Tenemos que modificar esta gramática para que se adhiera a las restricciones del teorema.

Para cada terminal  $x$  en  $G$ , introducimos un número no terminal único  $s$  y la producción  $s \mapsto x$ , luego reemplazamos las ocurrencias del terminal  $x$  en todas las demás reglas de  $G$  con  $s$ . Esto produce una gramática  $G'$  independiente del contexto en la cual el lado derecho de cada producción es una sólo terminal o una cadena de no terminales. Además  $L(G') = L(G)$ . Luego reemplazamos cada regla de  $G'$  de la forma:

$v \mapsto v_1, v_2, \dots, v_n$ , donde  $n > 2$ , con la colección de reglas

$$v \mapsto v_1 r_1$$

$$r_1 \mapsto v_2 r_2$$

$$r_{n-1} \mapsto v_{n-1} v_n$$

donde cada  $r_x$  es un no terminal único que no aparece en ningún otro lugar de la gramática. Obviamente, esta modificación no cambia el lenguaje generado por la gramática. Al llegar a esta etapa tenemos una gramática  $G''$  independiente del contexto que genera  $L$ , para la cual el lado derecho de cada producción es un sólo terminal, dos no terminales o un solo no terminal. Entonces, lo que queda por hacer es eliminar, las producciones de la última forma (un sólo no terminal). Para esto, consideramos cada secuencia de producciones de la forma:

$v_n \mapsto v_{n-1}, v_{n-1} \mapsto v_{n-2}, \dots, v_2 \mapsto v_1$ , introducimos regla  $v_n \mapsto x$  si  $v_1 \mapsto x$  es una producción de  $G''$ , e introducimos la producción  $v_n \mapsto pq$  si  $v_1 \mapsto pq$  está en  $G''$ . Una vez que se han agregado estas producciones, podemos eliminar aquellas donde el lado derecho contiene un sólo no terminal, sin reducir los poderes generativos de la gramática  $G''$ . Después de todo, cualquier derivación que usa las reglas de la forma  $u \mapsto v$  puede modificarse para que utilice en cambio las reglas que acabamos de introducir. Así, la gramática resultante genera  $L$  y satisface las restricciones del teorema.  $\square$

**Definición 4.2.** Se dice que una gramática independiente del contexto cuyas producciones se adhieren a las restricciones del teorema 2.1 tiene la forma normal de Chomsky.

## 4.2. Lista de Ejercicios

1. Construya un autómata de pila  $M$  para el cual

$$L(M) = \{w^r x^s y^t z^u : r + t = s + u\}$$

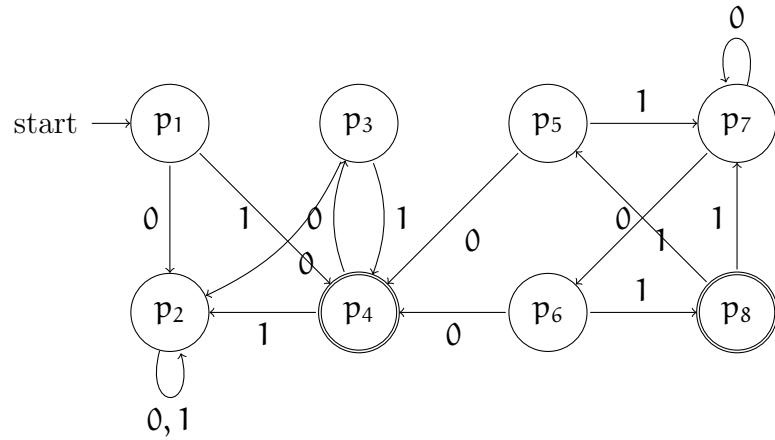
2. Construir autómatas finitos deterministas que reconozcan los siguientes lenguajes:

- $L_1 = \{a^m b^n | m > 0, n > 0\}$
- $L_2 = \{x \in \{0, 1\}^* | \text{en } x \text{ aparece el } 1 \text{ dos o tres veces, la primera y la segunda de las cuales no son consecutivas}\}$
- $L_3 = \{x \in \{a, b\}^* | N_a(x) \text{ es par}\}$
- $L_4 = \{x \in \{a, b\}^* | x \text{ acaba en } a\}$

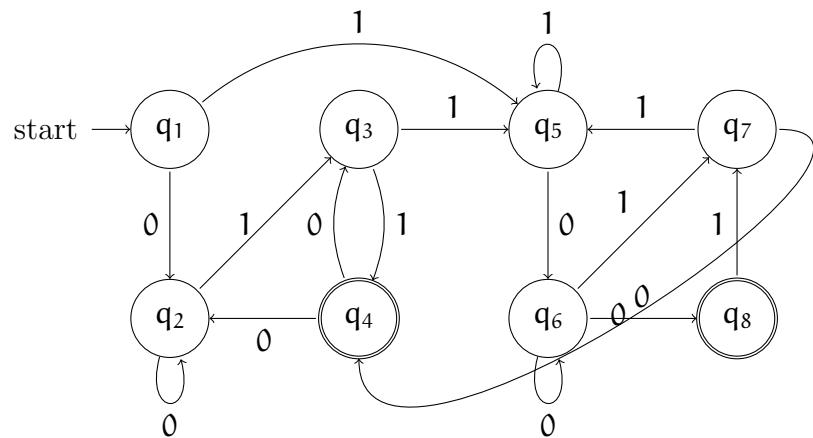
3. Construir un AFD mínimo que reconozca las palabras sobre el alfabeto  $\Sigma = \{a, b, c, d\}$  que contienen un número par (eventualmente cero) de apariciones de la subcadena  $bcd$ .

4. Dados los AF definidos por los siguientes diagramas de transición:

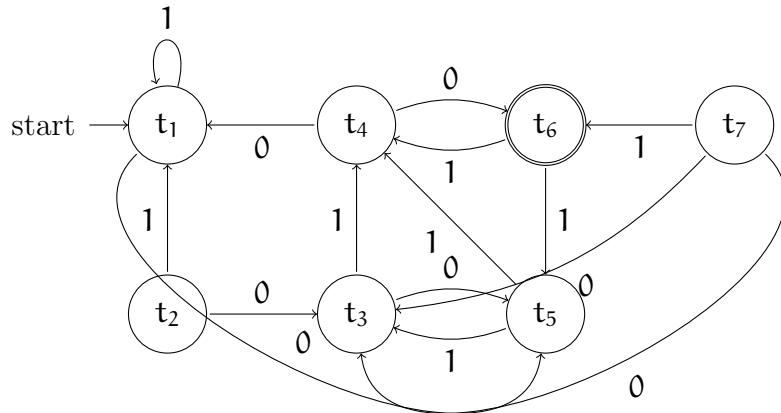
■  $A_p$



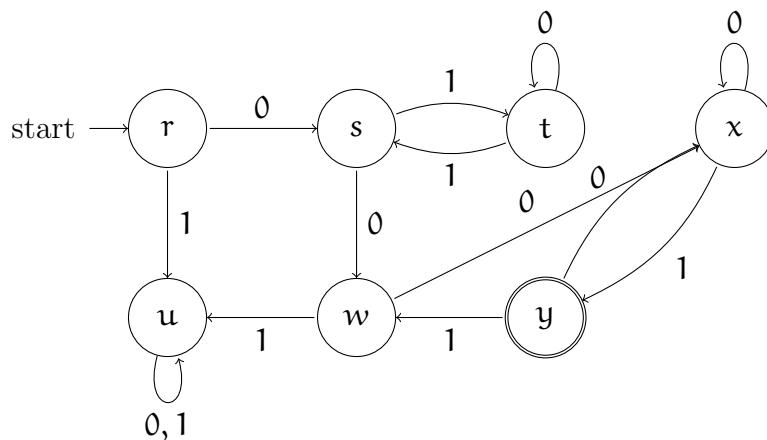
■  $A_q$



■  $A_t$



■  $A_r$

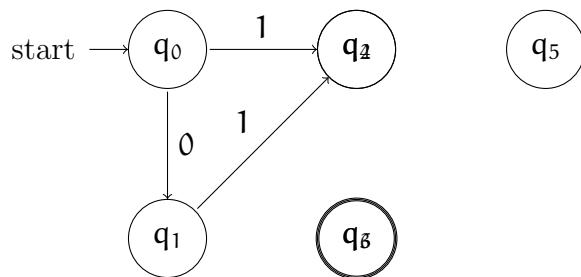


Obtener para cada uno de ellos el autómata mínimo.

Establecer si son o no equivalentes:

- Por suma directa de autómatas.
- Cuáles son isomorfos.

■ Dado el AFD A, obtener el autómata finito mínimo equivalente  $A'$



5. Convierta la siguiente gramática, con símbolo inicial  $S$ , en una gramática con forma normal Chomsky que genere el mismo lenguaje.

$$S \mapsto MzN$$

$$M \mapsto xM$$

$$M \mapsto \lambda$$

$$N \mapsto yN$$

$$N \mapsto \lambda$$

6. Demuestre que el lenguaje

$$L = \{x^n y^n z x^n y^n / n \in \mathbb{N}\}$$

no es independiente del contexto .

7. Construir un Automata de Pila que vacie su pila antes de aceptar cualquier cadena, cuyo lenguaje sea

$$L = \left\{ w_1 c w_2 / w_1, w_2 \{a, b\}^*, w_2 \neq w_1^{-1} \right\}$$

8. Convierta la siguiente gramática, con símbolo inicial  $S$ , en una gramática con forma normal Chomsky que genere el mismo lenguaje.

$$S \mapsto xSy$$

$$S \mapsto wNz$$

$$N \mapsto S$$

$$N \mapsto \lambda$$

9. Demuestre que el lenguaje  $L = \{x^n / n \text{ es primo}\}$

10. Dada la gramática independiente del contexto :

$$< S > ::= < A > < B > a < C >$$

$$< A > ::= < A > < B >$$

$$< B > ::= b | \lambda$$

$$< C > ::= < D > | \lambda$$

$$< D > ::= d$$

- a) Determine el lenguaje generado por esta gramática

- b) Determine la Forma Normal Chomsky que genere el mismo lenguaje que está gramática .

11. Decimos que una cadena de caracteres sobre el alfabeto  $\sum = \{[, ], (, )\}$  es correcta si todos los paréntesis y corchetes que se abren se cierran después, y ningún paréntesis o corchete se cierra dejando dentro otro sin cerrar.
  - a) Estudiar si el lenguaje formado por dichas cadenas de caracteres es independiente del contexto. En caso afirmativo, dar una gramática independiente del contexto y un autómata a pila para el mismo.
  - b) Hacer lo mismo con el lenguaje correspondiente de expresiones correctas que además de corchetes y paréntesis incluyen llaves abiertas y cerradas (“{” y “}”).
12. Demostrar que los siguientes lenguajes no son independientes del contexto:
  - a)  $L = \{a^n / n \text{ es primo}\}$
  - b)  $L = \{a^n b^j / n < j^2\}$
13. Decimos que una cadena de caracteres sobre el alfabeto  $\sum = \{[, ], (, )\}$  es correcta si todos los paréntesis y corchetes que se abren se cierran después, y ningún paréntesis o corchete se cierra dejando dentro otro sin cerrar. Estudiar si el lenguaje formado por dichas cadenas de caracteres es independiente del contexto.

# Capítulo 5

## Máquinas de turing

### 5.1. Conceptos

La máquina de Turing es un modelo computacional introducido por Alan Turing en el trabajo “On computable numbers, with an application to the Entscheidungsproblem”, publicado por la Sociedad Matemática de Londres, en el cual se estudiaba la cuestión planteada por David Hilbert sobre si las matemáticas son decidibles, es decir, si hay un método definido que pueda aplicarse a cualquier sentencia matemática y que nos diga si esa sentencia es cierta o no. Turing construyó un modelo formal de computador, la máquina de Turing, y demostró que existían problemas que una máquina no podía resolver. La máquina de Turing es un modelo matemático abstracto que formaliza el concepto de algoritmo.

Una máquina de Turing consta de:

1. Una unidad de control, que puede estar en cualquier estado
2. Una cinta dividida en casillas (cada casilla puede contener un símbolo)

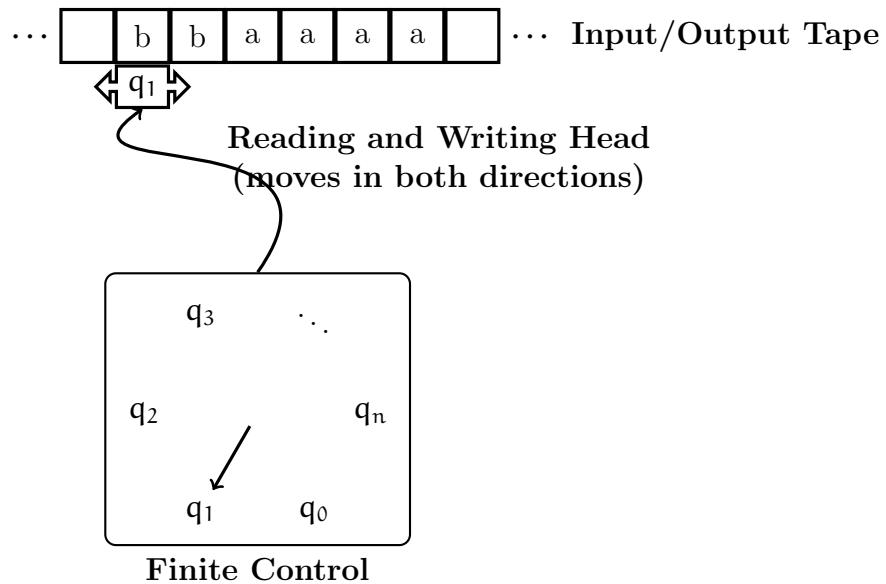


Figura 5.1: Máquina de turing

La entrada, una cadena de símbolos de longitud finita, elegidos del alfabeto de entrada, se encuentra en la cinta. El resto de las casillas, que se extiende infinitamente hacia ambos lados, contiene inicialmente un símbolo denominado espacio en blanco (es un símbolo de cinta, pero no de entrada). Existe una cabeza de cinta, que se encuentra al principio en la casilla de la entrada situada más a la izquierda. Un movimiento de la máquina de Turing es una función del estado y del símbolo de la cinta al que señala la cabeza. En un movimiento, la máquina de Turing:

1. Cambiará de estado (puede ser el mismo)
2. Escribirá un símbolo en la cinta (puede ser el mismo)
3. Moverá la cabeza hacia la izquierda o hacia la derecha. Se exige un movimiento, y no se permite que la cabeza permanezca en el mismo lugar. No es una limitación de la MT.

**Definición 5.1.** La máquina de Turing es un modelo matemático abstracto que formaliza el concepto de algoritmo. Una máquina de Turing con una sola

cinta puede ser definida como una 6-tupla.

$$\mathcal{M} = \{S, \Gamma, \Sigma, h, i, \delta\}$$

donde:

- $\Gamma$  es el alfabeto de símbolos de la cinta
- $\Sigma \subset \Gamma$  es el alfabeto de símbolos de entrada
- $h \in \Gamma$  es el símbolo en blanco  $h \notin \Sigma$
- $S$  es un conjunto finito de estados
- $i \in S$  es el estado inicial
- $\delta \subset S$  es el conjunto de estados finales
- $\delta$  es una función de transición parcial  $\delta$ :

$$S \times \Gamma \rightarrow S \times \Gamma \times \{I, D\}$$

En una transición, la máquina de Turing lee un símbolo de la cinta, cambia de estado y escribe un símbolo o efectúa un desplazamiento a izquierda o derecha.

**Ejemplo 5.1.1.** Así tenemos:

- $\delta(p, x) = (q, y)$  “  $p$  estado actual,  $x$  símbolo actual, ir al estado ‘ $q$ ’ escribiendo  $y$  sobre  $x$  ”
- $\delta(p, x) = (q, L)$  “  $p$  estado actual,  $x$  símbolo actual ir al estado ‘ $q$ ’ no viendo el cabezal  $L$  celda a la izquierda”

**Ejemplo 5.1.2.** Esta máquina solo se mueve hacia la derecha.

$$\Sigma = \{x, y\}$$

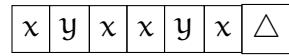


Figura 5.2: Máquina de turing

**Ejemplo 5.1.3.** Esta máquina se mueve a la derecha y busca una  $x$  y le quita  $x$ .

## 5.2. Construcción Modular de Máquinas de Turing

- Construcción de máquinas de Turing complejas a partir de bloques elementales.
- Transferencia de control entre máquinas :

$$\rightarrow M1 \xrightarrow{x} M2$$

- Transferencia de control con varios símbolos:

$$\rightarrow M1 \xrightarrow{\{x,y,z\}} M2 \xrightarrow{w} M3$$

### 5.2.1. Bloques de construcción básicos

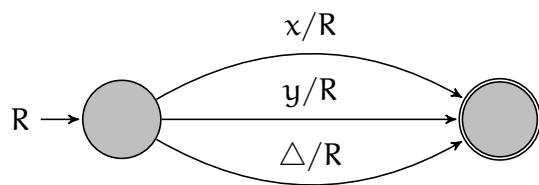


Figura 5.3: Funcionamiento de la Máquina de Turing

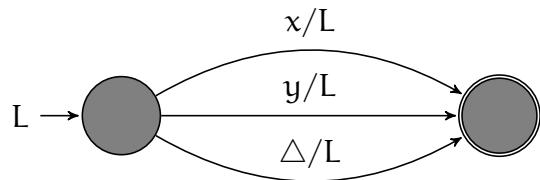


Figura 5.4: Funcionamiento de la Máquina de Turing

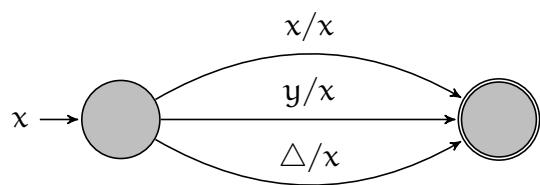


Figura 5.5: Funcionamiento de la Máquina de Turing

### 5.2.2. Bloques más complejos

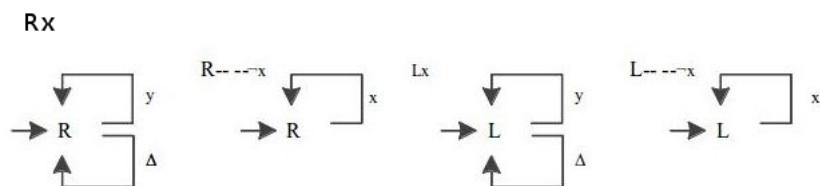


Figura 5.6: Funcionamiento de la Máquina de Turing

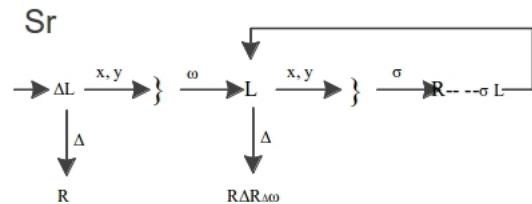


Figura 5.7: Funcionamiento de la Máquina de Turing

### 5.3. Evaluación de Cadenas

- Aceptación de una cadena
  - Situación inicial:
    - Cabeza en el extremo izquierdo de la cinta.
    - La cadena de entrada empieza en la segunda posición de la cinta.
- Situación final:
  - Máquina de Turing en estado de parada.
- Lenguaje aceptado por una máquina de Turing: conjunto de cadenas que acepta.
- Una máquina de Turing puede escribir un mensaje de aceptación si acepta una cadena.
  - Mensaje de aceptación:  $\Delta Y \Delta \Delta \Delta$
- Símbolos especiales :
  - #: extremo izquierdo de la cinta

- \*: extremo derecho de la porción alterada de la cinta
- Si la máquina es  $M$ , la máquina final es:

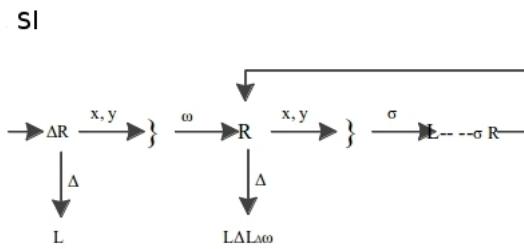


Figura 5.8: Funcionamiento de la Máquina de Turing

- $M_0$  es igual que  $M$ , salvo que:
  - Si  $M_0$  alcanza  $\#$ , se desplazará una casilla a la izquierda (terminación anormal)
  - Si  $M_0$  alcanza  $*$ , debe desplazarlo un lugar a la derecha ejecutando  $\rightarrow R * L\Delta$

### 5.3.1. Máquinas de Turing de varias cintas

- Cada cinta tiene su propia cabeza de lectura/escritura.
- Una transición:
  - Depende de los símbolos acutales de todas las cintas.
  - Sólo afecta a una cinta (escribir o desplazar).
- Estado inicial:
  - Contenido de la primera cinta:  $\triangleq w\Delta\Delta\Delta$
  - Las restantes cintas están en blanco, con la cabeza en su extremo izquierdo.

**Teorema 5.1.** Para cada máquina de Turing  $M$  de  $k$  cintas, existe una máquina de Turing  $M'$  con una cinta tal que:

$$L(M) = L(M')$$

*Demostración.* Construcción de  $M'$

- Previo:
  - Cada casilla en el conjunto de las  $k$  cintas se representa como una  $2k$ -tupla:
    - En la posición  $2n - 1$  de la tupla se tiene el símbolo de la cinta  $n$ .
    - En la posición  $2n$  de la tupla hay 1 si la cabeza está en la casilla,  $\Delta$  si no.
  - Es posible asignar un símbolo de cinta nuevo a cada posible  $2k$ -tupla.
    - Así se puede almacenar en una sola cinta toda la información del conjunto completo de cintas.
  - Cada estado de  $M'$  se representa como un *estado compuesto* formado por una  $k + 1$ -tupla cuyo primer elemento es el estado de la cinta, siendo los demás elementos los símbolos actuales de todas las cintas.
- Primer paso: traducir el contenido de la cinta de  $M'$  a un formato que represente todas las cintas de  $M$ .
  - Desplazar el contenido de la cinta a la derecha un lugar:
 
$$\rightarrow R_\Delta S_R L_\Delta$$
 (la cabeza queda en la segunda posición de la cinta).

- Mover la cabeza un lugar a la izquierda, escribir  $\#$  y mover un lugar a la derecha.
- Repetir los dos pasos siguientes hasta que el símbolo sustituido en  $b)$  sea un blanco:
  - a) Mover la cabeza un lugar a la derecha.

- b) Sustituir el simbolo actual  $x$  por la tupla  $(x, \Delta, \Delta, \dots, \Delta)$ .
- Ejecutar  $L_\Delta$ , lo que pondria la cabeza en la segunda casilla de la cinta, y escribir en ella la tupla  $(\Delta, 1, \Delta, 1, \dots, \Delta, 1)$ .
  - Simulación de  $M'$ 
    - Cada transición de  $M$  supone una secuencia de pasos en  $M'$ .
    - Estado inicial:  $(i, \Delta, \Delta, \dots, \Delta)$ .
    - Una transicion  $(\tau)$  solo afecta a una cinta  $(j_\tau)$ .
    - Secuencia correspondiente a la transicion  $\tau$ :
      - Mover la cabeza a la derecha hasta que el componente  $2j$  de la  $2k$ -tupla sea 1.
      - Si la transicion es una escritura, modificar el componente  $2j_\tau - 1$  de la tupla.
      - Si la transición produce un movimiento a la derecha:
        - ◊ Reemplazar el componente  $2j_\tau - 1$  por un  $\Delta$ .
        - ◊ Mover la cabeza a la derecha.
        - ◊ Si encuentra un  $\Delta$ , sustituirlo por la  $2k$ -tupla  $(\Delta, \Delta, \dots, \Delta)$ .
        - ◊ Reemplazar el componente  $j_\tau - 1$  por un 1.
      - Si la transición produce un movimiento a la izquierda:
        - ◊ Reemplazar el componente  $2j_\tau - 1$  por un  $\Delta$ .
        - ◊ Mover la cabeza a la izquierda.
        - ◊ Si encuentra un  $\#$ , mover a la izquierda (terminación anormal).
        - ◊ Reemplazar el componente  $2j_\tau - 1$  por un 1.
    - Colocar la cabeza en la segunda posición de la cinta (después de buscar a la izquierda el símbolo  $\#$ ), y pasar al nuevo estado compuesto de  $M'$ .
  - La máquina  $M'$  así construida simula  $M$ , y acepta el mismo lenguaje que ella.

□

## 5.4. Máquinas de Turing no deterministas

- El no determinismo puede consistir en que:
  - La máquina no se encuentre totalmente definida.
  - Se ofrezcan alternativas en algunos pares estado-símbolo.
- $M(S, \Sigma, \Gamma, \pi, i, h)$  donde  $\pi$  es un subconjunto de
 
$$((S - h) \times \Gamma) \times (S \times (\Gamma \cup L, R))$$
- Una máquina de Turing no determinista acepta una cadena  $w$  cuando es posible que llegue al estado de parada después de iniciar sus cálculos con la entrada  $w$ .

**Teorema 5.2.** *Para toda máquina de Turing  $M$  no determinista existe una máquina determinista  $D$  que acepta el mismo lenguaje que  $M$ .*

*Demostración.* .

- Para toda máquina de Turing  $M$  no determinista existe una máquina determinista  $M'$  con tres cintas que acepta el mismo lenguaje.
  - 1<sup>a</sup> cinta: contiene la cadena de entrada.
  - 2<sup>a</sup> cinta: cinta de trabajo. En ella se copia la cinta 1 (desplazada un lugar a la derecha), marcando el principio de la cinta con  $\#$  y marcando el final de la entrada con  $*$ . En este cinta se simulan secuencias de transiciones.
  - 3<sup>a</sup> cinta: control de las transiciones efectuadas.
- Simulación de  $M$ :
  1. Copiar la cadena de entrada de 1 a 2 (con los marcadores de inicio y fin).
  2. Generar la siguiente secuencia de transiciones en la cinta 3.
  3. Simular la secuencia con la cinta 2.
  4. Si se llega al estado de parada de  $M$ , detenerse. Si no, borrar la cinta 2 y volver al paso 1.

□

## 5.5. Lenguajes aceptados por Máquinas de Turing

- Gramáticas estructuradas por frases:
  - Parte izquierda de las reglas: combinación de símbolos terminales y no terminales, con al menos un no terminal.
  - Parte derecha de las reglas: combinación de símbolos terminales y no terminales de cualquier longitud (incluso 0).
- Las máquinas de Turing aceptan lenguajes estructurados por frases.
- Configuración de una máquina de Turing:
  - Contenido de la cinta: entre corchetes.
  - A la izquierda del símbolo actual se incluye el estado.
- Aceptacion: secuencia de configuraciones de la maquina que empieza con  $[i\Delta w\Delta]$  y termina con  $[[h\Delta Y\Delta]]$

**Teorema 5.3.** *Todo lenguaje aceptado por una máquina de Turing es un lenguaje estructurado por frases.*

*Demostración.* La gramática sera:

$$(V, \Sigma, S, R)$$

donde

- $V = S, [,], \Delta, Y \cup \Gamma$
- $\Sigma =$ alfabeto de M
- $S:$  axioma.

■ R:

$$S \rightarrow [h\Delta Y \Delta]$$

$$\Delta \rightarrow \Delta \Delta$$

$\delta(p, x) = (q, y)$  produce la regla  $qy \rightarrow px$

$\delta(p, x) = (q, R)$  produce la regla  $xq \rightarrow px$

$\delta(p, x) = (q, L)$  produce las reglas  $qyx \rightarrow ypx \forall y \in \Gamma$

$$i\Delta \rightarrow \lambda$$

$$\Delta \Delta \rightarrow \Delta$$

$$\Delta \rightarrow \lambda$$

□

**Teorema 5.4.** *Todo lenguaje estructurado por frases es aceptado por una máquina de Turing.*

*Demostración.* Para cada gramática  $G$  existe una máquina de Turing no determinista  $M$  de 2 cintas que acepta el lenguaje generado por  $G$ .

Construcción de la máquina:

1. Se copia la cadena de entrada en la primera cinta.
2. Se escribe  $S$  (símbolo inicial) en la cinta 2.
3. Se aplican las reglas de reescritura de forma no determinista a la cadena de la cinta 2.
4. Si la cinta 2 contiene sólo símbolos terminales, se compara con la cadena de la cinta 1. Si son iguales, el proceso ha terminado. Si no, provocar una terminación anormal.

□

**Teorema 5.5.** *Dado un alfabeto  $\Sigma$  existe al menos un lenguaje  $L$  definido sobre  $\Sigma$  que no es un lenguaje estructurado por frases.*

### Sistemas de codificación de máquinas de Turing

- *Cada estado se representa como una cadena de ceros.*
  - $i = 0$
  - $h = 00$
  - *Los demás: a partir de 000 en adelante.*
- *Cada simbolo de  $\Sigma$ , así como L y R, se representan como cadenas de ceros.*
  - $L = 0$
  - $R = 00$
  - *Simbolos de  $\Sigma$ : a partir de 000 en adelante.*
  - *Espacio en blanco: cadena vacía.*
- *Cada transición se puede representar como un conjunto de cadenas de ceros separadas cada una por un único 1.*
- *El conjunto de transiciones de la máquina se representa como la secuencia de cadenas de ceros y unos de todas sus transiciones, poniendo un 1 al principio del todo y un 1 al final, y un solo 1 para separar dos transiciones consecutivas.*

#### 5.5.1. Máquinas de Turing universales

- Son máquinas de Turing programables que pueden simular a cualquier otra máquina de Turing.
  - Programa: máquina de Turing simulada codificada + cinta de entrada (con un 1 al principio y un 1 al final).
- Constan de 3 cintas:

- 1<sup>a</sup> cinta: programa + cadena de entrada.  
\* Contendrá la salida de la máquina.
- 2<sup>a</sup> cinta: área de trabajo (manipulación de datos).
- 3<sup>a</sup> cinta: estado actual de la máquina simulada.
  
- La máquina universal:
  - Copia la cadena de entrada de la cinta 1 a la cinta 2.
  - Graba el código del estado inicial en la cinta 3.
  - Busca una transición aplicable en la máquina codificada de la cinta 1. Cuando la encuentra:
    - Realiza la transición en la cinta 2.
    - Escribe el nuevo estado en la cinta 3.
  - La máquina universal continúa con este proceso hasta que llega al estado de parada de la máquina simulada. Entonces copia la cinta 2 en la cinta 1, coloca la cabeza de la cinta 1 donde se encontraba la de la cinta 2 y se detiene.

### 5.5.2. Lenguajes Aceptables y Decidibles

- Lenguaje decidable: es aquel lenguaje  $L$  para el cual existe una máquina de Turing que puede aceptar cualquier cadena  $w \in L$  y rechazar cualquier cadena  $w \notin L$ .
- Lenguaje aceptable: es aquel lenguaje  $L$  para el cual no existe ninguna máquina de Turing que pueda aceptar cualquier cadena  $w \in L$  y rechazar cualquier cadena  $w \notin L$ .
- Lenguajes recursivamente enumerables: lenguajes estructurados por frases.
- Lenguajes recursivos: lenguajes decidibles por una máquina de Turing.

### 5.5.3. El Problema de la Parada

- Maquina autotermiante: maquina con un alfabeto  $0, 1, \Delta$  que se detiene cuando se le mete como entrada una cadena que es ella misma codificada.
- Existen maquinas de Turing para las que no es posible decidir si son autotermiantes o no: es decir, no puede saberse con certeza si se detienen  $\Rightarrow$  no puede saberse con certeza si una maquina de Turing se va a detener ante una cadena de entrada  $\Rightarrow$  EL PROBLEMA DE PARADA NO ES DECIDIBLE.

## 5.6. Lista de Ejercicios

1. Presenta un argumento de que cualquier lenguaje que contenga un número finito de cadenas es decidable por máquinas de Turing .
2. Dibuje el diagrama de transiciones para la máquina de Turing compuesta  $\rightarrow R_x \Delta L$ .
3. Calcule  $eq(5,3)$  y  $eq(5,5)$  de acuerdo con las siguientes definiciones :  
 $eq(x,y) = 1 \ominus ((y \ominus x) + (x \ominus y))$   
 $monus(x,0) = \pi_1^1(x)$   
 $monus(x,y+1) = h(x,y,monus(x,y))$  ,  $h(x,y,z) = pred(z)$   
 $pred(0) = \sigma()$  ,  $pred(y+1) = \pi_1^2(y,pred(y))$   
 $monus(x,y) = x \ominus y$
4. Sea  $\Sigma$  el alfabeto de la lengua castellana . Construir una máquina de Turing que con la cabeza situada en el símbolo más a la izquierda de una palabra  $w \in \Sigma^*$  escriba , a la izquierda de  $w$  y dejando un blanco de separación , la longitud de la palabra  $w$ , expresada como un número en sistema binario. La palabra  $w$ , al finalizar el proceso, deberá mantenerse en el lugar que ocupaba inicialmente. Por ejemplo :  
Configuración inicial :  $\Delta turing$  , configuración final :  $\Delta 110 \Delta turing$   
.
5. Multiplicador unario. Dados dos números  $x$  e  $y$  sobre el alfabeto  $\Sigma = \{1\}$ , diseñar una máquina de Turing que los multiplique. Los contenidos inicial y final de la cinta son los siguientes:

La cabeza se encuentra inicialmente sobre el primer blanco a la derecha del número  $x$ .

6. Sea el alfabeto de la lengua castellana

$$\Sigma = \{a, b, c, \dots, k, l, m, n, \dots, r, s, \dots, x, y, z\}$$

- a) Construir una máquina de Turing que con la cabeza situada en el símbolo más a la izquierda de una palabra  $w \in \Sigma^*$  escriba, a la izquierda de  $w$  y dejando un blanco de separación, la longitud de la palabra  $w$ , expresada como un número en sistema binario. La palabra  $w$ , al finalizar el proceso, deberá mantenerse en el lugar que ocupaba inicialmente.
  - b) Escribir el proceso que realiza la máquina de Turing construida para la palabra  $w = \text{turing}$ . Se considerará también la palabra vacía, que pertenece a  $\Sigma^*$  y cuya longitud es cero.
7. a) Construir una Máquina de Turing tal que colocada en el dígito más a la izquierda de un número  $w$  expresado en unario ( $w \in \{1\}^*$ ) y distinto de cero, lo copia a la derecha del número dado, sin dejar ninguna separación entre el original y la copia. Probar el funcionamiento con  $w = 1111$ .
- b) Construir una Máquina de Turing tal que colocada en el dígito más a la izquierda de un número natural  $n$  expresado en unario ( $n \in \{1\}^*$ ) y distinto de cero, calcula  $2^n$ , también expresado en unario, y lo coloca a la derecha del número dado  $n$ , dejando un blanco de separación. Probar el funcionamiento con  $n = 3$ , i.e.,  $n = 111$ . (Indicación:  $2^n$  en unario es igual a  $2^{n-1}$  en unario seguido de  $2^{n-1}$  en unario:  $11111111 = 23 = 2222 = 11111111$ ).
8. Construir una máquina de Turing cuya cinta en la situación inicial contiene  $n$  números en unario, separados por asteriscos, y en la situación final deja la cinta con los mismos números en unario ordenados de menor a mayor y de izquierda a derecha y sobre el mismo espacio de la cinta en que se encontraban los números inicialmente.
9. a) Construir una máquina de Turing que transforma un número entero expresado en sistema unario en el correspondiente número en sistema decimal. Dicha máquina, con la cabeza situada en el dígito más a la izquierda de una palabra sobre el alfabeto  $\{a\}$ , escribirá a la izquierda de este número y dejando un blanco de separación, el correspondiente número decimal (una palabra sobre el alfabeto

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ) . Notas: 1.- Para el número en unario se utiliza el símbolo a para evitar confusiones. 2.- El número unario debe mantenerse en su posición y tal como estaba al principio. (Indicación: por cada dígito del número unario que se va recorriendo se aumenta en una unidad el número decimal correspondiente).

- b) Construir una máquina de Turing que realiza el mismo proceso que en el apartado a) pero colocando el número decimal a la **derecha** del número unario, dejando un blanco de separación.
  - c) Construir una máquina de Turing que transforme un número decimal en un número unario. Con la cabeza situada en el dígito más a la izquierda de un número decimal, destruye éste y escribe el correspondiente número en sistema unario. (Indicación: por cada dígito del número unario que se va añadiendo se resta una unidad al número decimal).
  - d) Diseñar una máquina que realiza la misma tarea que en el apartado c), pero dejando el número decimal tal y como estaba y en su mismo sitio y el número unario a la derecha y separado por un blanco.
10. Sea  $n$  un número unario tal que  $n = 2^x$ ,  $x \geq 1$ . Construir una máquina de Turing que, con la cabeza situada inicialmente en el dígito más a la izquierda del número  $n$ , escriba a la derecha de  $n$ , y dejando una celda en blanco intermedia, el valor en unario de  $\log_2 n$ . Indicación: Se aconseja utilizar la relación recursiva  $\log_2 2n = \log_2 n + 1$ .  $\log_2 2 = 1$
11. Construir una Máquina de Turing que decida el predicado mayor  $(x, y)$ , donde  $x$  e  $y$  son números naturales mayores que cero, expresados en unario.
- Es decir que, dados dos números  $x$  e  $y$ , escriba a continuación de ellos y dejando un blanco de separación un 1, si  $x > y$ , y un 0 en caso contrario.
12. Construir sendas máquinas de Turing que reconozcan los siguientes lenguajes:
- a)  $\{a^n b^{2n} | n > 0\}$
  - b)  $\{ww | w \in \{a, b\}^*\}$
  - c)  $L = \{x \in \{0, 1\}^* | N_0(x) = 2N_1(x)\}$
13. Diseña una maquina de Turing que calcule la “division entera de dos numeros enteros positivos”.

14. Dibuje el diagrama de transiciones para la maquina de Turing compuesta  $\rightarrow R_x \Delta L$ .
15. Diséñese una máquina de Turing, que acepte el lenguaje, dentro del alfabeto {a, b, c}, cuyas cadenas no tienen ninguna a en su primera mitad y ninguna b en su segunda. A tenor de esta especificación, debe entenderse que las palabras de longitud impar no son admitidas.
16. Muestre que el lenguaje

$$L = \{ a^{n^2} / n \geq 0 \}$$

es aceptado por máquinas de Turing. ¿Es decidible? ¿Por qué?

17. Dibuje el diagrama de transiciones para la máquina de Turing compuesta

$$\rightarrow R_x \Delta L$$

.

18. ¿Forma la unión de un número finito de lenguajes estructurados por frases un lenguaje estructurado por frases? . Justifique su respuesta.
19. Presente un argumento de que cualquier lenguaje que contenga un número finito de cadenas es decidible por máquinas de Turing .
20. Codifique la Máquina de Turing  $L_{\neg y} \Delta R$
21. ¿ El lenguaje que acepta exactamente aquellas cadenas de ceros y unos que sean representaciones válidas de Máquinas de Turing codificadas, es decidible ?

# Índice de figuras

1.1.	Pintura idealizada de Euclides.	6
1.2.	Kurt Gödel.	7
1.3.	ábaco romano.	8
1.4.	Pascalina en el Museo de Artes y Oficios de París.	9
1.5.	La máquina analítica de Babbage, como se puede apreciar en el Science Museum de Londres.	9
1.6.	El telar de Charles Jacquard.	10
1.7.	Mark I, fotografía de 1948.	11
1.8.	Glen Beck (background) and Betty Snyder (foreground) program ENIAC in BRL building 328. (U.S. Army photo).	12
1.9.	Computadora EDVAC.	13
1.10.	von Neumann en 1940.	14
1.11.	UNIVAC I en Franklin Life Insurance Company.	15
1.12.	IBM 650.	16
1.13.	IUNIVAC 1105 operator console, in front of the cabinets containing the CPU and memory.	17
1.14.	The University of Manchester Atlas in January 1963.	18
1.15.	WordStar corriendo en DOS.	19
1.16.	RCA-601.	20
1.17.	IBM-360.	21
1.18.	CDC-7600.	22
1.19.	System /370-145.	23
1.20.	Steve Jobs y Steve Wozniak.	24
3.1.	Autómata Finito Determinista	46

3.2. Diagrama de transición que acepta cadenas que representan enteros o cadenas que representan números reales en notación decimal . . . . .	51
3.3. Diagrama de transición . . . . .	52
4.1. . . . .	60
4.2. Transiciones ( $p, x, s; q, y$ ) . . . . .	61
4.3. Diagrama de transición para $M = (S, \Sigma, \Gamma, \rho, s_0, F)$ . . . . .	62
4.4. Diagrama de transición para $M = (S, \Sigma, \Gamma, \rho, s_0, F)$ . . . . .	63
4.5. Diagrama de transición para $M = (S, \Sigma, \Gamma, \rho, s_0, F)$ . . . . .	64
4.6. Pila con elementos $x, y, z$ . . . . .	66
4.7. Autómata de pila . . . . .	67
4.8. Autómata de pila . . . . .	69
5.1. Máquina de turing . . . . .	82
5.2. Máquina de turing . . . . .	84
5.3. Funcionamiento de la Máquina de Turing . . . . .	84
5.4. Funcionamiento de la Máquina de Turing . . . . .	85
5.5. Funcionamiento de la Máquina de Turing . . . . .	85
5.6. Funcionamiento de la Máquina de Turing . . . . .	85
5.7. Funcionamiento de la Máquina de Turing . . . . .	86
5.8. Funcionamiento de la Máquina de Turing . . . . .	87

# Bibliografía

[Hopcroft, Motwani, Ullman, 2007] *Teoría de autómatas, Lenguajes y Computación*, Addison Wesley Iberoamericana, USA

[J. Glenn Brookshear, 1993] *Teoría de la Computación Lenguajes Formales, autómatas y Complejidad*, Addison Wesley Iberoamericana, USA