APROXIMACIÓN AL ENTSCHEIDUNGSPROBLEM DESDE LA TEORÍA DE AUTÓMATAS Y MÁQUINAS DE TURING

Cristian Camilo Castellanos Camargo
1024487111-2008240015
Yerson Libardo Díaz Suárez
80854701-2008240027

Universidad Pedagógica Nacional
Facultad de Ciencia y Tecnología
Licenciatura en Matemáticas
Bogotá
2015

APROXIMACIÓN AL ENTSCHEIDUNGSPROBLEM DESDE LA TEORÍA DE AUTÓMATAS Y MÁQUINAS DE TURING

Cristian Camilo Castellanos Camargo
1024487111-2008240015
Yerson Libardo Díaz Suárez
80854701-2008240027

Modalidad: Trabajo de grado asociado a un tema específico de investigación para optar al título de Licenciado en matemáticas

Asesor
Jorge Edgar Páez, Msc. Matemáticas

Universidad Pedagógica Nacional
Facultad de Ciencia y Tecnología
Licenciatura en Matemáticas
Bogotá
2015

Dedicatoria

Para Dios en primer lugar.
Para mis padres y hermanos.
Cristian

A mi padre y a mi madre por su ejemplo.
A Luis Carlos y Jonathan por su amistad y compañía
y a Fanny Andrea por su apoyo y amor incondicional.
Yerson

Agradecimientos

Un profundo agradecimiento a Dios por darnos la inspiración,
a nuestros padres y hermanos por su incondicional apoyo,
a nuestros maestros por sus enseñanzas
y a nuestro asesor por su confianza y dedicación.
Cristian

A todos los maestros que he tenido la oportunidad de conocer
e igualmente a nuestro asesor por su confianza y apoyo.
Yerson.

1. Información General	
Tipo de documento	Trabajo de grado
Acceso al documento	Universidad Pedagógica Nacional. Biblioteca Central
Título del documento	Aproximación al Entscheidungsproblem desde la Teoría de Autómatas y Máquinas de Turing
Autor(es)	Castellanos Camargo, Cristian Camilo; Díaz Suárez, Yerson Libardo
Director	Páez , Jorge Edgar
Publicación	Bogotá. Universidad Pedagógica Nacional, 2015, 120 p.
Unidad Patrocinante	Universidad Pedagógica Nacional
Palabras Claves	ENTSCHEIDUNGSPROBLEM; MÁQUINA DE TURING; ALGORITMO; AUTÓMATA; PROBLEMA INDECIDIBLE; LENGUAJE Y PROBLEMA DE LA PARADA

2. Descripción

Trabajo de grado que se propone para el estudio de la teoría de la computación o análisis del Entscheidungsproblem, por medio de la definición de máquinas de Turing (MT); en el cual se encontrará una breve reseña histórica desde Leibniz hasta Turing del desarrollo de la noción de algoritmo.

Se definen y clasifican los autómatas finitos para entender el funcionamiento, definición, caracterización y simulación de las Máquinas de Turing, con el fin de reconocer los lenguajes recursivamente enumerables que son recursivos, los cuales son equivalentes a una definición de decibilidad y serán de gran importancia para la aproximación a la demostración del Entscheidungsproblem.

3. Fuentes

Apostol, T. (2001). *Calculus* (Segunda edición., Vols. 1-2, Vol. 2). Barcelona: Reverté.

Beuchot, M. (1986). La conceptografía y la lógica formal de Frege. *Revista Elementos, Ciencia y Cultura*, *2*(9), 70-75.

Beuchot, M. (2008). *Introducción a la lógica*. México, D.F: UNAM.

Beuchot, M. (2011). *Manual de Filosofía*. México, D.F: Ediciones Paulinas.

Boole, G. (1854). *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. New York: Dover Publications.

Cadevall, S. (1976). Decibilidad de la Lógica de Predicados Monádicos por el Método de la Recusación. *Teorema: Revista Internacional de Filosofía*, *6*(3-4), 455-484.

Davis, M. (2002). *La computadora universal. De Leibniz a Turing*. Barcelona: Debate.

De Castro, R. (2004). *Teoría de la computación. Lenguajes, autómatas, gramáticas* (Primera edición.). Bogotá: Universidad Nacional de Colombia.

Gallardo, D., Arques, P., & Lesta, I. (2003). *Introducción a la teoría de la computabilidad* (Segunda Edición.). Alicante: Publicaciones Universidad de Alicante.

García, L., & Martínez, G. (2005). *Apuntes de Teoría de Autómatas y lenguajes formales*. Recuperado a partir de http://repositori.uji.es/xmlui/handle/10234/5995?locale-attribute=es

Hernández, F., & Morado, R. (2007). Hilbert, Turing y la Noción de Procedimiento Efectivo. En *Sociedad y ciencia* (pp. 313-320). México D.F: Siglo XXI editores.

Holcombe, M. (1982). *Algebraic Automata Theory*. Cambridge University Press.

Martínez, P. (1985). El origen histórico de la lógica matemática: Boole. *Thémata: Revista de filosofía*, (2), 87 - 98.

Meléndez, R. (1992). Observaciones sobre la demostración de Gödel. En P. Gómez & C. Gómez, *Sistemas Formales, informalmente ¿Por qué intentaron formalizar a la*

matemática si era tan buena muchacha? (Segunda edición., pp. 109-111). Bogotá: Una empresa docente/Universidad de los Andes. Recuperado a partir de http://funes.uniandes.edu.co/668/1/Gomez1999Sistemas.pdf

Muñoz, J. (2002). *Introducción a la teoría de conjuntos* (Cuarta edición.). Bogotá: Universidad Nacional de Colombia.

Navarrete, I., Cárdenas, M., Sánchez, D., Botía, J., Marín, R., & Martínez, R. (2003). *Teoría de autómatas y lenguajes formales*. Universidad de Murcia. Recuperado a partir de http://tux.uis.edu.co/lenguajes/doc/Murcia.pdf

Pérez, J., & Caparrini, F. (2003). *Máquinas moleculares basadas en ADN*. Universidad de Sevilla.

Wise, H. (1996). *Breaking the Code*. Documental, biografía, British Broadcasting Corporation (BBC). Recuperado a partir de https://www.youtube.com/watch?v=S23yie-779k

4. Contenidos

Reseña histórica de la evolución del concepto de máquina de Turing (Algoritmo) y su relación con la teoría de la computación, la cual se establece definiendo y caracterizando los autómatas finitos y los lenguajes que son aceptados por estos, con el fin de comprender más fácilmente el funcionamiento, definición, clasificación y simulación de las máquinas Turing (MT).

Al final se establece la tesis de Church–Turing para sustentar nuestra aproximación a una posible demostración del Entscheidungsproblem por medio del problema de la parada.

5. Metodología

Para el desarrollo de este trabajo de grado, no se utilizó ninguna herramienta de recolección de datos.

6. Conclusiones

- La demostración de la indecibilidad del Problema de la Parada, por medio de las Máquinas de Turing, permite establecer una solución negativa al Entscheidungsproblem. Sin embargo, llegar a esta conclusión implica admitir que la noción de algoritmo es equivalente al modelo propuesto por Alan Turing, es decir que se acepta la tesis de

Church-Turing.

- La indecibilidad de la lógica de primer orden está relacionada con la estructura misma del sistema. Por ejemplo, la Aritmética de Peano a pesar de trabajar con el conjunto numerable de los números naturales y con finitos símbolos implica la existencia de funciones no computables relacionadas con la no enumerabilidad.
- El poder computacional de las Máquinas de Turing es limitado; es por ello que a pesar de que hay una conexión intuitiva entre la noción de algoritmo y la Máquina de Turing no se descarta la posibilidad de que la tesis de Church-Turing pueda ser un día refutada.
- Una solución para el Entscheidungsproblem según Hilbert debe cumplir con los presupuestos establecidos por la noción de procedimiento efectivo. Se encontró, a partir de la definición de una Máquina de Turing, los requerimientos de Hilbert son satisfechos de manera parcial pues no es posible que una MT establezca a priori los pasos necesarios para computar una cadena.
- El estudio de los lenguajes regulares y la teoría de Autómatas, previo al de las máquinas de Turing, permite conocer con mayor facilidad las propiedades computacionales de los lenguajes aceptados por una Máquina de Turing.
- La diferencia significativa entre las MT y los autómatas es el tipo de movimientos del cabezal, además de los dominios y rangos de las funciones de transición.
- El lenguaje aceptado por un autómata es recursivamente enumerable, pues cualquier autómata puede ser simulado usando las máquinas de Turing. Empero, no toda Máquina de Turing se puede representar con autómatas.

Elaborado por:	Castellanos Camargo, Cristian Camilo; Díaz Suárez, Yerson Libardo
Revisado por:	Páez , Jorge Edgar

Fecha de elaboración del Resumen:	28	01	2015

Contenido

Lista de Tablas

Lista de Ilustraciones

Glosario

AUTÓMATA: máquina de reconocimiento de lenguajes que posee un conjunto finito de estados y lee símbolos escritos en una cinta por medio de una unidad de control o cabezal.

CONECTIVOS LÓGICOS: símbolos que se utilizan para formar proposiciones a partir de otras proposiciones; algunos de estos son: conjunción $(\wedge)$, disyunción $(\vee)$, implicación $(\Rightarrow)$, doble implicación $(\Leftrightarrow)$, etc.

ENTSCHEIDUNGSPROBLEM: vocablo alemán que designa al Problema de la Decisión propuesto por David Hilbert en 1928.

LENGUAJE: conjunto de cadenas (o palabras) compuestas a su vez por caracteres o símbolos.

LÓGICA: ciencia y arte que fundamenta las leyes, modos y formas de pensamiento[1]. Más precisamente, la simbología que usa la lógica determina las reglas que gobiernan los razonamientos matemáticos (Beuchot, 2008, p. 117).

LÓGICA DE PRIMER ORDEN: también conocida como lógica de predicados. Es utilizada para expresar el significado de un amplio dominio de proposiciones. Un predicado es una sentencia que contiene una o más variables y se convierte en una proposición cuando las variables son sustituidas por constantes. Por ejemplo: $P(x) = x^2 \leq 20$ es un predicado con dominio en los números naturales, donde $P(-2)$ es verdadera y $P(-6)$ es falsa.

LÓGICA PROPOSICIONAL: aquella que está compuesta por conectivos lógicos y proposiciones. Representa hechos mediante sentencias las cuales pueden ser verdaderas o falsas, pero no ambas al mismo tiempo.

MÁQUINA DE TURING (MT): máquina con la capacidad de manipular caracteres en una cinta. Los caracteres pueden ser símbolos alfa-numéricos. La máquina de Turing

[1] Esta definición superficial de lógica está basada en Beuchot (2011, pp. 37-39) y (2008, p. 14).

es la máquina que manipula caracteres con entera independencia de cualquier significado que ellos tengan.

PROPOSICIÓN: es toda oración de la que se puede decidir si es verdadera o falsa.

RAZONAMIENTO DEDUCTIVO: un razonamiento es deductivo si y sólo si las premisas son la evidencia de la verdad de la conclusión.

REGLAS DE INFERENCIA: comprende todo esquema válido de razonamiento, por ejemplo Ley de Modus Ponens, Ley Modus Tolens y Ley del Silogismo Hipotético.

SÍMBOLO: en teoría de la computación es un carácter que pertenece a un alfabeto.

TEOREMA: esquema válido de razonamiento donde el conjunto de premisas se denomina hipótesis y la conclusión tesis.

UNIDAD DE CONTROL (UC): en un autómata corresponde al cabezal que lee los símbolos de una cinta finita o infinita y que de acuerdo a la función de transición este puede cambiar de posición ya sea a la izquierda o a la derecha. Existen casos en los que la unidad de control no cambia de posición.

Resumen

El presente trabajo tiene como objetivo exponer cómo Alan Turing consigue demostrar que el Entscheidungsproblem (problema de la decisión) no tiene solución. Gracias al diseño de una máquina abstracta de cómputo conocida como la máquina de Turing y al planteamiento del problema de la parada, se muestra que no existe un procedimiento mecánico o algoritmo que permita conocer a priori si un problema es o no demostrable o si una proposición es verdadera o falsa.

El estudio de la decibilidad de un problema tiene como fin dar respuesta a la necesidad histórica de disponer de un lenguaje universal con el fin de expresar cualquier idea y mecanizar cualquier tipo de razonamiento matemático. Es por ello que David Hilbert a principios del siglo XX con el objetivo de reducir las matemáticas a la manipulación formal de símbolos plantea de manera concreta tres preguntas que inquirían acerca de la completitud, la consistencia y la decibilidad de un sistema formal o axiomático.

Para abordar el problema de la decisión se comienza con el estudio de los lenguajes y los distintos dispositivos que los aceptan (Autómatas y Máquinas de Turing), según la clasificación realizada por el lingüista Noam Chomsky. De esta manera se logra caracterizar los recursos computacionales que exigen ciertos problemas los cuales se han clasificado como decidibles o indecidibles con el fin de dar respuesta negativa al Entscheidungsproblem.

Palabras clave: ENTSCHEIDUNGSPROBLEM; PROBEMA DE LA DECISIÓN; COMPUTACIÓN; MÁQUINA DE TURING; PROBLEMA DE LA PARADA; ALGORITMO; DECIBILIDAD; INDECIBILIDAD; AUTÓMATA; PROBLEMA INDECIDIBLE y LENGUAJE.

1. Introducción

Cuando Hilbert propuso el Entscheidungsproblem en 1900, se dio inició una nueva vía de investigación que condujo a Alonzo Church y a Alan Turing en 1936 y 1935 respectivamente a mostrar la imposibilidad de resolverlo. De este modo nace formalmente la Teoría de la Computación, rama de las matemáticas que se encarga básicamente del estudio de los problemas de decisión.

El estudio de los de Autómatas y sus lenguajes asociados muestra las razones por las cuales la Máquina de Turing es considerada en la actualidad como el instrumento con la mayor capacidad de cómputo para ejecutar algoritmos (tesis de Church-Turing).

El final de este recorrido culmina una sencilla demostración de que el problema de la parada es indecidible o que no tiene solución pues no existe una Máquina de Turing o un lenguaje con los recursos computacionales exigidos para resolverlo. De este modo se da una respuesta negativa al Entscheidungsproblem.

2. Justificación

El presente trabajo pretende conocer cómo Alan Turing con la creación de un dispositivo mecánico abstracto dio respuesta negativa al Entscheidungsproblem propuesto por David Hilbert y las razones por las cuales la máquina de Turing es considerada como el artefacto con el mayor poder computacional para ejecutar procedimientos de cálculo.

Para comprender el problema de la decisión es necesario conocer elementos básicos de la teoría de los autómatas y las máquinas de Turing. De este modo se presenta el problema de la parada, también propuesto por Alan Turing, el cual permite establecer las razones por las cuales no existe un algoritmo de algoritmos, dando así respuesta al Entscheidungsproblem.

3. Objetivo General

Elaborar una monografía que recopile información relevante de la Teoría de Autómatas, Máquinas de Turing y su relación con el Entscheidungsproblem teniendo en cuenta la tesis de Church–Turing.

Objetivos específicos

- Estudiar de manera inductiva los lenguajes relacionados con la tipificación realizada por Noam Chomsky de acuerdo a los recursos computacionales y las máquinas que los aceptan.
- Sintetizar los elementos conceptuales para comprender el funcionamiento de los Autómatas y las Máquinas de Turing, su equivalencia y los lenguajes asociados.
- Comprender la tesis de Church–Turing mostrando las razones por las cuales la comunidad científica considera que existe una conexión directa e intuitiva de la Máquina de Turing y la noción de algoritmo.
- Realizar una aproximación al Entscheidungsproblem, mostrando por medio del problema de la parada las razones por las cuales el problema de la decisión no tiene solución.
- Realizar una breve comparación entre las exigencias del Entscheidungsproblem, propuesto por Hilbert y las limitaciones de las Máquinas de Turing respecto a su capacidad para ejecutar algoritmos.

4. Marco Histórico

4.1 La idea de Leibniz

Gottfried Wilhelm Von Leibniz (1646-1716), tomando sus propios desarrollos matemáticos respecto a la notación y relación[2] del cálculo integral y diferencial[3] que permitían realizar cálculos complicados de forma sencilla, creía en la posibilidad de desarrollar un código de conceptos, donde llegaría de forma fácil y exacta a la verdad de las proposiciones y relaciones lógicas que sustentaría su idea, según Davis (2002, p. 16), de construir máquinas capaces de realizar cálculos difíciles con el fin de liberar la mente humana para el pensamiento creativo. Al desarrollar esta propuesta, Leibniz presenta formalmente un modelo perfeccionado de la máquina de Pascal en 1673 que introducía la multiplicación y la división; un año después construye otra capaz de resolver ecuaciones algebraicas y durante este proceso descubre la notación binaria que usaría para reconocer algunas propiedades de los números naturales. Davis (2002, pp. 30-31) menciona que la empresa de Leibniz se fundamentaba en tres acciones:

i. Seleccionar símbolos adecuados para generar un compendio de conocimientos que abarque la totalidad del conocimiento humano (lenguaje científico universal).
ii. Discriminar los conceptos claves subyacentes y dotar a cada uno de ellos de los símbolos adecuados (teoría formal de la definición).
iii. Reducir a unas reglas de deducción la manipulación de los símbolos (cálculo razonador o teoría formal de la deducción).

Esto último Leibniz lo denomino *calculusratiocinator* o lo que se conoce actualmente como lógica simbólica[4] y que según Martínez (1985, p. 88) permaneció inédita hasta principios del siglo XX y es una anticipación de la lógica matemática ya que se propuso

[2] Está relación se conoce como el teorema fundamental del cálculo.
[3] En el desarrollo cálculo integral (estudio del área bajo una curva limitada) presenta la notación $\int$ y del cálculo diferencial (estudio de las razones de cambio) presenta la notación d, las cuales son usadas aún en nuestros días.
[4] Para Beuchot (2008, p. 117) por lógica simbólica se entiende un desarrollo actual de la lógica formal utilizando el simbolismo convencional y una metodología rigurosa. Se le nomina simbólica por su trato con símbolos. Para efectos prácticos el símbolo no permite que se le interprete irregularmente sino que está algo sujeto a reglas.

un algebra de la lógica, con reglas para manipular conceptos lógicos, es decir, una teoría que ofrecería unos principios y una metodología para resolver problemas críticos.

4.2 Lógica y algebra de Boole

George Boole (1815-1864) concibió la lógica simbólica de Leibniz que Aristóteles había introducido dos mil años antes. Él pretendía expresar relaciones lógicas en la forma del algebra actual donde los símbolos expresan cantidades y las operaciones obedecen ciertas reglas y leyes básicas.

Para hacernos una idea de la obra de Boole, Davis (2002, p. 41) menciona que "aplicó métodos algebraicos a objetos que los matemáticos llaman operadores. Estos <<operan>> sobre las expresiones del álgebra ordinaria para dar lugar a nuevas expresiones". Así Boole reconoce la importancia de utilizar letras para representar una clase o grupo[5] en el razonamiento lógico, un artificio utilizado en el álgebra ordinaria para representar cantidades.

El álgebra de Boole aplicada a la lógica reconoce un resultado importante. Si x e y son dos clases, el producto xy representa la actual intersección de conjuntos, por tanto $xx = x$. En el álgebra de la secundaria $xx = x$ sucede cuando $x = 1$ ó $x = 0$, luego para que esto tenga sentido en el álgebra de la lógica de Boole era necesario reinterpretar los símbolos 0 y 1 como clases. Haciendo uso del producto entre clases se definieron las siguientes nociones básicas:

- "0" como la clase que no tiene elementos (conjunto vacío) y $0x = 0$.
- "1" como la clase a la que pertenece cualquier objeto en consideración (conjunto referencial) y se cumple que $1x = x$.

Del mismo modo haciendo uso de los signos de adición y sustracción usuales Boole definió la unión y la diferencia entre conjuntos respectivamente y construyó la

[5] Conjunto de todos los objetos descritos por una palabra en cuestión. Por ejemplo x es clase de seres vertebrados.

demostración de que nada puede pertenecer y no pertenecer al mismo tiempo a una clase determinada, lo que no lo llevaría a una contradicción[6].

Un esbozo y muestra del resultado anterior es el siguiente: Boole se refirió a $x - y$ como a las cosas que están en x pero que a su vez no están en y. Asimismo, $x + y$ representa la clase de todas las cosas que contienen a x e y. Así se deduce que $x + (1 - x) = 1$. También, si $x = xx = x^2$, entonces $x - x^2 = 0$; factorizando esta última expresión $x(1 - x) = 0$. En conclusión, nada puede pertenecer y no pertenecer al mismo tiempo a una clase x determinada.

El gran aporte de Boole manifiesta que la deducción lógica podía intuirse como una rama de las matemáticas (en particular del álgebra) por medio del empleo de símbolos. Desde allí la lógica ha sido objeto de ininterrumpidos avances a tal punto de darle su forma actual. Martínez (1985, p. 89) afirma: "Boole desarrolló una lógica usando los recursos del álgebra ordinaria pero que no es un álgebra matemática".

4.3 La conceptografía de Frege

Gottlob Frege (1848-1925) a diferencia de Boole tenía la intención de sustentar la matemática en la lógica[7]. Para esto Frege quería desarrollar una lógica sin el uso de la misma en el proceso (Davis, 2002, p. 70) y a su vez explicar la formación de nuevas conexiones y conceptos para tratar los objetos que tiene como universo de discurso (Beuchot, 1986, p. 71). Nace entonces la noción de conceptografía o como la denomina Beuchot "*escritura conceptual*" como un lenguaje artificial con unas reglas gramaticales (más precisamente de sintaxis) estrictas. El fin de este proyecto era presentar las inferencias lógicas como operaciones puramente mecánicas.

Frege quién había aplicado métodos lógicos a la fundamentación de la aritmética, creó el primer sistema lógico que encerraba todo el razonamiento que usaban los

[6] Principio de no contradicción definido por Aristóteles como el axioma fundamental de toda la filosofía: ''una cosa no puede ser y no ser al mismo tiempo y bajo la misma circunstancia'' (Beuchot, 2011, p. 38).
[7] En otras palabras Frege quería que las operaciones matemáticas fuesen reducibles a las operaciones lógicas.

matemáticos y los filósofos, en él destacaba el uso de proposiciones secundarias[8], conectores lógicos, cuantificadores, variables, constantes, entre otros (Beuchot, 1986, p. 73). Sin embargo las reglas de Frege no proporcionaban un medio para verificar la conclusión buscada sino sólo hasta cuando esta era demostrada, es decir, no se proporcionaba la manera de saber si la conclusión se podía o no deducir de las premisas dadas. De este modo *la idea de Leibniz* no se satisfacía por completo puesto que este lenguaje no era un instrumento de cálculo eficaz.

En conclusión, las desventajas del método de Frege consistían en lo complicada que se podía convertir una simple deducción ya que no se proporcionó ningún procedimiento de cálculo en su conceptografía (Davis, 2002, p. 75). Posteriormente, en 1936 se demostró que no existe un método general que mostrara *a priori* si una inferencia era correcta o no en un sistema lógico o axiomático deductivo, siendo esto lo que lleva a Turing a formular la idea de una computadora de propósito general[9].

A pesar de los esfuerzos de Frege su lógica era muy complicada e inconsistente como se lo hizo saber en una carta Bertrand Russell en 1902 con la llamada la *paradoja de Russell*[10]. Sin embargo dada la inconsistencia, Frege trato de contra-argumentar sin éxito.

Por último cabe señalar que la importancia de Frege en este manuscrito radica en que él estableció el primer lenguaje formal universal artificial. Davis (2002, p. 70) al respecto menciona que "la conceptografía era el antecesor de todos los lenguajes de programación informáticos que se usan comúnmente en la actualidad".

[8] Para George Boole las proposiciones secundarias son aquellas que se refieren o relacionan con otras proposiciones consideradas como verdaderas o falsas (Boole, 1854, p. 160).

[9] Las computadoras de propósito general son aquellas que se pueden programar para un sin número de requerimientos o necesidades.

[10] Russell dividió los conjuntos en aquellos que pertenecen a sí mismos y los que no. Por ejemplo el conjunto de todas las "cosas" que no son números naturales, no es un número natural, por ello este conjunto se pertenece a sí mismo. La paradoja de Russell considera al conjunto $\boldsymbol{R}$ de los conjuntos que no se pertenecen a sí mismos, es decir $\boldsymbol{R} = \{X | X \notin X\}$. Lo anterior conduce a cuestionar si $\boldsymbol{R} \in \boldsymbol{R}$**.** Si la respuesta es afirmativa es porque $\boldsymbol{R}$ verifica la propiedad que define a $\boldsymbol{R}$, es decir $\boldsymbol{R} \notin \boldsymbol{R}$**.** Si la respuesta es negativa, entonces por definición $\boldsymbol{R} \in \boldsymbol{R}$. En cualquier caso se llega a que $\boldsymbol{R} \in \boldsymbol{R} \leftrightarrow \boldsymbol{R} \notin \boldsymbol{R}$. Russell llegó a la conclusión que $\boldsymbol{R}$ no puede existir.

4.4 Los números transfinitos de Cantor y el método de la diagonalización

George Cantor (1845-1918) inició su investigación sobre series infinitas asociadas a las funciones trigonométricas y esto lo llevo a estudiar las clases de infinito existentes en los conjuntos numéricos. Cantor demostró que la cantidad de elementos del conjunto de los naturales es el mismo que el de los pares usando el siguiente emparejamiento (Ilustración 4.1) ya propuesto anteriormente por Leibniz (Davis, 2002, p. 83):

2	4	6	8	…
↑	↑	↑	↑	…
1	2	3	4	…

Ilustración 4.1 Emparejamiento de los números naturales y pares

Sin embargo, Leibniz a diferencia de Cantor y basado en la sentencia de Euclides “el todo es mayor que las parte” negó el concepto del *número de los* números naturales. A pesar de la resistencia de la comunidad académica de la época, Cantor decide estudiar una teoría aplicable a los conjuntos infinitos y aceptó que un conjunto infinito pueda tener el mismo número de elementos de una de sus partes (Davis, 2002, p. 84).

Posteriormente y obteniendo el mismo resultado, Cantor trabajó el conjunto de las fracciones positivas y que a simple vista tendría un número de elementos mayor que el de los naturales. Cantor realiza una agrupación de los números racionales con los naturales teniendo en cuenta que la suma del denominador con el numerador sea igual a un número natural. La Ilustración 4.2 muestra la fracción cuyo denominador y numerador sumados es dos, luego las fracciones cuya suma es tres y así sucesivamente.

$$\frac{1}{1};\quad \frac{1}{2},\frac{2}{1};\quad \frac{1}{3},\frac{2}{2},\frac{3}{1};\quad \frac{1}{4},\frac{2}{3},\frac{3}{2},\frac{4}{1};\quad \frac{1}{5},\frac{2}{4},\frac{3}{3},\frac{4}{2},\frac{5}{1};\quad \dots$$

Ilustración 4.2 Fracciones positivas ordenadas según la suma del denominador y el numerador

De esta forma pudo emparejar todos los números racionales con los naturales de la siguiente manera (Ilustración 4.3):

$\frac{1}{1}$	$\frac{1}{2}$	$\frac{2}{1}$	$\frac{1}{3}$	$\frac{2}{2}$	$\frac{3}{1}$	$\frac{1}{4}$	$\frac{2}{3}$	$\frac{3}{2}$	$\frac{4}{1}$	…
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	…
1	2	3	4	5	6	7	8	9	10	…

Ilustración 4.3 Emparejamiento de las fracciones positivas con los números naturales

El anterior razonamiento podría llevar a suponer que los números reales también tendrán una correspondencia uno a uno con los naturales y por lo tanto compartirán el mismo número de elementos. Sin embargo, Cantor demostró que el conjunto de los números reales no puede emparejarse con el de los números naturales.

Cantor llamó transfinitos a los cardinales de los conjuntos infinitos y al cardinal del conjunto de los números naturales lo denomino $\aleph_0$ (Aleph sub cero). Cantor utilizó el símbolo $\mathfrak{C}$ para representar el cardinal del conjunto de los reales. Él estaba seguro de que $\mathfrak{C}$ es el siguiente número cardinal después de $\aleph_0$. Esta se conoce como la *hipótesis del continuo* y a pesar de los intentos, Cantor no demostraría está conjetura.

En 1891 Cantor publicó un artículo en el que presentaba su método de diagonalización el cual le proporcionó una forma de reconocer que hay más números reales que naturales. Para ilustrar este procedimiento se usa el siguiente ejemplo propuesto por Davis (2002, pp. 94-97): en una baraja de cartas francesa existen cuatro palos de mazos: ♣, ♦, ♥, ♠. Las letras A, B, C, D son etiquetas o marcas de los conjuntos $A = \{♦, ♥\}, B = \{♦, ♠\}, C = \{♣, ♥\},\ D = \{♣, ♦\}$. Relacionando la información anterior en la Tabla 4.1 el símbolo (+) menciona si el elemento está en determinado conjunto y (–) si no.

Tabla 4.1 Mazos y cartas

	♣	♦	♥	♠
A	−	+	+	−
B	−	+	−	+
C	+	−	+	−
D	+	+	−	−

El método de la diagonal es una técnica para combinar los elementos de varios conjuntos en un conjunto nuevo E, que será diferente a los demás. Empero, antes de nominar los elementos del conjunto E se cambia el valor de los resultados obtenidos en la diagonal principal de la Tabla 4.1, como se muestra en la Tabla 4.2.

Tabla 4.2 Nuevo conjunto *E*

	♣	♦	♥	♠
E	+	−	−	+

Se obtiene un nuevo conjunto $E = \{\clubsuit, \spadesuit\}$ que es diferente a cualquiera de los anteriores etiquetados. Este procedimiento es válido si utilizamos en vez de cuatro mazos los números naturales y emparejamos igual cantidad de subconjuntos (finitos o infinitos de números naturales). Con este procedimiento se llegará a un nuevo conjunto distinto de los subconjuntos propuestos, por tanto ningún emparejamiento de este estilo podrá contener a todos los números naturales ya que el conjunto nuevo no está asociado a algún número natural. Así el cardinal de todos los subconjuntos de los números naturales es mayor que $\aleph_0$[11] y es posible mostrar que este número no es otro que $\mathfrak{C}$ (Davis, 2002, pp. 97, 253). Con este método también se puede obtener un número mayor que $\mathfrak{C}$. Sin embargo, razonar de esta manera podía llevar a resultados paradójicos, puesto que si hay un conjunto de todos los números cardinales ¿cuál debe ser su número cardinal? Resultaba que este conjunto tendría que ser mayor que cualquier número cardinal y ¿Cómo un número cardinal podía ser mayor que todos los números cardinales?

Posteriormente Bertrand Russell, quién cuestionó las ideas de Frege y ahora también las de Cantor al lanzar la pregunta: ¿puede haber un conjunto de todos los conjuntos?, si existiera ¿qué pasaría si se empaquetaran conjuntos arbitrarios y utilizar luego conjuntos para etiquetar los paquetes?[12]. Aunque estas preguntas no dependían de consideraciones de números transfinitos se creó un ambiente de desconfianza hacia la teoría de conjuntos que opacaría su avance por algún tiempo.

[11] Teorema de cantor $A \prec \mathcal{P}(A)$ (Muñoz, 2002, p. 237)
[12] Paradoja de Bertrand Russell, conjunto de todos aquellos conjuntos que no pertenecen a sí mismos.

4.5 Hilbert y el *Entscheidungsproblem*

David Hilbert (1862-1943) destacado por sus asombrosos logros en el campo del álgebra presenta en París en el año 1900 en el congreso internacional de matemáticas veintitrés problemas que parecían completamente inabordables. El primer problema de la lista era el de decidir la verdad o falsedad de la hipótesis del continuo de cantor. El segundo problema requería probar que los axiomas de la aritmética de los números reales son consistentes. Respecto a lo último, Bertrand Russell trabajó incesantemente para desarrollar un sistema de lógica simbólica con el fin de reducir la aritmética a la lógica (empresa que inició Frege) haciéndole frente a las paradojas. Para Hilbert la contradicción que aparecía al suponer la existencia de un conjunto formado por todos los números cardinales transfinitos de Cantor mostraba simplemente que tal conjunto no existía.

El esfuerzo de Russell y sus seguidores por formalizar la matemática produjo reacciones por parte de los contradictores, uno de ellos Jules Henri Poincaré. Davis (2002, p. 114) cita textualmente las palabras de Poincaré y que muestran la posibilidad de reducir la matemática a la computación de manera irónica:

> Así se comprenderá rápidamente que para demostrar un teorema ya no sea necesario, ni siquiera útil, saber qué significa […] podríamos imaginarnos una máquina en la que introdujéramos axiomas por un extremo y obtuviéramos teoremas por otro, igual que aquella legendaria máquina de Chicago en la que los cerdos entraban vivos y salían convertidos en jamones y salchichas. Ya no es necesario que los matemáticos tengan más conocimiento de lo que están haciendo aquellas máquinas.

Al respaldo de los contradictores de Russell y Hilbert se encontraba el matemático holandés llamado L.E.J. Brouwer y mencionaba al respecto (Davis, 2002, p. 117): “de algunas proposiciones no pueden decirse ni que sean ciertas ni que sean falsas; son proposiciones con las que no hay ningún método conocido mediante el cual se pueda decidir una cosa u otra”, esta afirmación criticaba el uso del principio del tercio excluso.

Volviendo a Hilbert, su programa exigía que toda derivación en un sistema formal debía ser consecuencia de aplicar reglas de manera finita, a esto se le llamó métodos

finitarios (Davis, 2002, p. 121). De esta manera, surgieron dos problemas (Davis, 2002, p. 123):

> Uno de estos problemas era demostrar que la lógica de primer orden es *completa* en el sentido de que cualquier fórmula que desde fuera se considera válida puede deducirse desde dentro del sistema utilizando sólo las reglas que se proponían en los libros de texto. El segundo, que acabó conociéndose como el Entscheidungsproblem de Hilbert, era el de proporcionar un método según el cual, dada una fórmula de la lógica de primer orden, se pudiera determinar en un número finito de pasos concretos y bien definidos si dicha fórmula es válida.

El primer problema sobre la completitud se puede dividir en dos cuestiones referidas a si las matemáticas son completas y consistentes; un tercer punto inquiere si son decidibles o computables, en últimas se quería indagar sobre:

i. Completitud: ¿Son completas las matemáticas? Es decir ¿se puede probar que una proposición es verdadera o falsa sin ninguna duda?
ii. Consistencia: ¿Son consistentes las matemáticas? O de otro modo ¿ocurrirá que una proposición es verdadera o falsa a la vez?
iii. Decibilidad (Entscheidungsproblem): ¿Las matemáticas son computables o decidibles? En otras palabras ¿Se puede diseñar un algoritmo que partiendo de una proposición y un número finito de pasos diga si una conclusión es demostrable?

Hilbert en su manual de lógica de 1928 sostuvo (a pesar de plantearlo en forma de pregunta) que no existen fisuras en las reglas de las inferencias deductivas que permitían que validar las conclusiones obtenidas de las premisas, sin embargo él quería una demostración (Davis, 2002, p. 134). Por tanto las esperanzas de Leibniz aún se mantenían.

4.6 Gödel desmorona los planes

Kurt Gödel (1906-1978) en 1931 consiguió en su tesis doctoral mostrar (*teorema de la Completitud*) que el cálculo de predicados es totalmente completo, es decir que cada predicado válido es cierto o falso utilizando métodos finitarios, tal como lo exigía Hilbert, sin embargo, en 1936 y a partir de este trabajo Gödel pudo demostrar (*Teorema de la Incompletitud*) que un sistema axiomático no es completo mencionando que no se pueden contener todos los enunciados verdaderos de la teoría que se pretende formalizar (García & Martínez, 2005, p. 108).

El teorema de Incompletud en términos del teorema de la de la Completitud dice que no toda inferencia válida tenía asociada una demostración paso a paso de la conclusión partiendo de premisas y usando las reglas de Frege y Russell, es decir que hay proposiciones de las que no se puede deducir sin son falsas o verdaderas dentro de un sistema axiomático. Los métodos expuestos por Gödel no eran precisamente novedosos, sin embargo el teorema de incompletud no se puede demostrar sin utilizar métodos no finitarios. Gödel no encontró razón para aceptar la limitación propuesta por Hilbert (Davis, 2002, p. 138).

Davis (2002, p. 141) intenta explicar el otro hallazgo de Gödel respecto a la consistencia de las matemáticas, mencionando:

> Gödel descubrió que hay proposiciones que, vistas desde el exterior de esos sistemas, podían considerarse verdaderas, aunque no pudieran demostrarse desde dentro de los mismos […] Gödel llegó a la sorprendente conclusión de que, muy al contrario, no sólo hay una noción de verdad matemática plena de sentido, sino que también su alcance se extiende más allá de lo que se puede demostrar en cualquier sistema formal dado. Esta conclusión se aplicaba a una amplia gama de sistemas lógico-formales […] Para cualquiera de estos sistemas, existen proposiciones verdaderas que pueden expresarse en dicho sistema pero que no se pueden demostrar en el seno del mismo […] mostrando así que ni los sistemas lógicos más poderosos podían albergar esperanza alguna de abarcar toda la esfera de verdad matemática.

Cuando Hilbert plantea axiomas para la Aritmética de manera que todo enunciado finitista *verdadero* sea *demostrable* y para cualquier otro enunciado su verdad o

falsedad sea demostrable, lo que quería decir era que si un enunciado es "demostrable" entonces "verdadero". Lo que prueba Gödel es que en la Aritmética de Peano: "verdad" y "demostrabilidad" no es lo mismo[13]. En general, dado un conjunto de axiomas siempre existirá algún enunciado E donde tanto él como su negación no son demostrables, por lo que este E quedaría fuera de la definición de "verdad". Consideremos el siguiente ejemplo: dados los axiomas $A_1, \dots, A_n$ y el enunciado $0 + 1 = 0$ demostrable a partir de estos. No se puede suponer que $A_1, \dots, A_n$ es un sistema inconsistente, de hecho estos axiomas pueden formar un sistema consistente, en tanto la noción de demostrable es diferente de la de verdadero ya que usualmente $0 + 1 = 1$. Ahora, demostrar que $0 + 1 = 0$ implica la existencia de una sucesión finita de enunciados cuyo enunciado final es $0 + 1 = 0$ por aplicación del Modus Ponens. Por tanto el concepto de verdad se diluye en este caso.

Por otro lado, si los axiomas $A_1, \dots, A_n$ cumplieran con que todo enunciado finitista verdadero es demostrable, tendríamos posiblemente un sistema inconsistente, ya que la negación de $0 + 1 = 0$ es un enunciado verdadero con otro sistema de axiomas y no demostrable a partir de $A_1, \dots, A_n$.

A pesar de los resultados de Gödel, Hilbert aún esperaba una solución para el Entscheidungsproblem.

4.7 Alan Turing y la teoría de la computación

Alan Mathison Turing nació el 23 de Junio de 1912 en Londres. La lectura en 1932 del trabajo de John Von Newman (1903-1957) sobre los fundamentos lógicos de Mecánica Cuántica le ayudó a Turing ser un inflexible investigador intelectual del King´s College en Cambridge (Davis, 2002, p. 170,171). En 1935, Alan Turing asiste a una clase impartida por Newman sobre los fundamentos de las matemáticas y se entera del Entscheidungsproblem propuesto por Hilbert. A partir de ese momento Turing empieza a trabajar en la demostración de que no existe un algoritmo de tales características.

[13] Una explicación a grandes rasgos en tres páginas de la demostración del teorema de Gödel se encuentra en Meléndez (me 1992, pp. 109-111).

Con resultados similares (aunque con aproximaciones un tanto divergentes) y en países distintos, Alonso Church (Estados Unidos) y Alan Turing (Reino Unido) responden negativamente al Entscheidungsproblem o tercer problema formulado por Hilbert, sin embargo, la propuesta presentada por Turing es más sencilla.

Esta propuesta salta a la luz en un artículo publicado por Alan Turing en 1936 llamado *On the Computable Numbers with a application Entscheidungsproblem* basándose en el concepto de algoritmo[14] lo cual es equivalente a aquello que puede realizar una Máquina de Turing con el fin de determinar si una conclusión particular puede derivarse de ciertas premisas con el uso de ciertas reglas. Concretamente Turing traduce los resultados de Gödel al lenguaje de la computación por medio de una codificación binaria. Es así como la máquina de Turing se convierte en la principal herramienta del estudio de la computabilidad.

La Máquina de Turing es un instrumento abstracto conformado por una unidad de control que se mueve de un estado a otro siguiendo un número finito de reglas concretas, también puede escribir un símbolo en una cinta (infinita) o borrarlo. Estas características le brindan la capacidad a la Máquina de Turing de realizar cualquier computación matemática si esta se presenta por medio de un algoritmo. Con esta máquina Turing mostró que no hay solución al Entscheidungsproblem demostrando que el problema de la parada es indecidible para una Máquina de Turing, a grosso modo, esto es equivalente a decir que no es posible decidir algorítmicamente si una Máquina de Turín se detendrá.

De este modo Alan Turing con la propuesta de una máquina universal (máquina universal de Turing), capaz de realizar el trabajo de cualquier otra máquina, traza el diseño del computador actual y ayuda a comprender que es lo que se puede esperar del software.

El 7 de junio de 1954 en Wilmslow y por razones aún no muy claras Alan Turing se suicidó consumiendo una manzana sumergida previamente en cianuro luego de ser judicialmente procesado por su condición homosexual en el Reino Unido.

[14] Esta propuesta Turing la desarrolla sin existir en lo época una definición rigurosa de algoritmo.

5. Preliminares

Con el fin de introducir a la teoría de Autómatas y las Máquinas de Turing, se presentan a continuación algunas definiciones necesarias en el desarrollo de la teoría de la computación.

5.1 Alfabetos, Cadenas y Lenguajes

Definición 5.1 Un **alfabeto** es un conjunto no vacío finito cuyos elementos se denominan símbolos y se denota con la letra Σ.

Ejemplo 5.1 $\Sigma = \{0,1\}$ se conoce como el alfabeto binario compuesto por los símbolos 0 y 1.

Definición 5.2 Una **cadena** u sobre un alfabeto Σ es cualquier sucesión finita de elementos que pertenecen a Σ. Se acepta la existencia de una cadena que no tiene símbolos denominada cadena vacía y se denota con λ.

Definición 5.3 **El conjunto de todas las cadenas** sobre un alfabeto Σ, incluyendo la cadena vacía, se denota por Σ^*.

Ejemplo 5.2 Sea $\Sigma = \{c, d\}$ el alfabeto que consta de los símbolos c y d. Las siguientes son cadenas que están en Σ^*: $cdc,\ cdcdddc,\ ccccddd,$ etc. Considérese que $cdc \neq ccd$, pues el orden de la cadena es relevante ya que las cadenas se definen como sucesiones[15]. En términos generales se tiene que:

$$\Sigma^* = \{\lambda, c, d, cc, cd, dc, dd, ccc, ccd, cdc, cdd, dcc, dcd, ddc, ddd, \dots\}.$$

Definición 5.4 La **longitud de una cadena** $v \in \Sigma^*$ es el número de símbolos que componen la cadena y se denota $|v|$.

[15] Conjuntos secuencialmente ordenados.

Ejemplo 5.3 Sea $\Sigma = \{a, b, c, d\}$ el alfabeto que consta de los símbolos a, b, c y d. La longitud de cada una de las siguientes cadenas que están en Σ^* es:

$$|w| = |abadc| = 5$$

$$|x| = |ababadcdcb| = 10$$

$$|y| = |aaabbbcddcc| = 11$$

$$|\lambda| = 0$$

Definición 5.5 Dado un alfabeto Σ y $u, v \in \Sigma^*$, la **concatenación** de dos cadenas u, v se denotan como uv y:

1. Si $v = \lambda$, entonces $u\lambda = \lambda u = u$.
2. Si $u = a_1 \dots a_n$ y $v = b_1 \dots b_m$, entonces $uv = a_1 \dots a_n b_1 \dots b_m$ donde $n, m \in \mathbb{N}$[16].

Ejemplo 5.4 Sea $\Sigma = \{a, b, c, d\}$ y dos cadenas $u = abcdabcd$, $v = ccddbbaa$ que pertenecen a Σ^*. La concatenación uv es: $uv = abcdabcdccddbbaa$.

Teorema 5.1 La concatenación de cadenas es una operación asociativa. Es decir si $u, v, w \in \Sigma^*$, entonces $(uv)w = u(vw)$.

Demostración: **s**ea $u = a_1 \dots a_n$, $v = b_1 \dots b_m$ y $w = c_1 \dots c_p$ tres cadenas sobre Σ^*. Aplicando dos veces la Definición 5.5**:**

$$(uv)w = (a_1 \dots a_n b_1 \dots b_m)c_1 \dots c_p = a_1 \dots a_n b_1 \dots b_m c_1 \dots c_p$$

Ahora nuevamente por la Definición 5.5 la siguiente cadena la separamos así:

$$a_1 \dots a_n b_1 \dots b_m c_1 \dots c_p = a_1 \dots a_n (b_1 \dots b_m c_1 \dots c_p)$$

Esta última cadena no es más que $u(vw)$ ■

Definición 5.6 Un **prefijo** v de una cadena u es tal que $u = vw$ para alguna cadena w de Σ^*.

[16] $\mathbb{N} = \{x: x\ es\ un\ número\ natural\}$

Definición 5.7 Un **sufijo** v de una cadena u es tal que $u = wv$ para alguna cadena w de Σ^*.

Ejemplo 5.5 Sea $\Sigma = \{a, b, c, d\}$ y $u = acbdb$, los prefijos[17] de u son: $\lambda, a, ac, acb, acbd, acbdb$. Los sufijos de u son: $\lambda, b, db, bdb, cbdb, \ acbdb$.

Definición 5.8 Un **lenguaje** sobre un alfabeto Σ es un subconjunto de Σ^*, se denota usualmente por L, en otros términos $L \subseteq \Sigma^*$.

Ejemplo 5.6 Sea $\Sigma = \{a, b, d\}$. $L = \{ab, bd, abad\}$ es un lenguaje y $L \subseteq \Sigma^*$.

Nota 5.1 Puesto que para todo lenguaje L se tiene que $L \subseteq \Sigma^*$, las siguientes son propiedades conjuntistas entre lenguajes, unión: $A \cup B$, intersección $B \cap A$, diferencia de conjuntos $A - B$ y complemento $A^C = \Sigma^* - A$, donde A, B son lenguajes sobre Σ.

Definición 5.9 La **concatenación de dos lenguajes** A y B sobre Σ se denota AB y se define como $AB = \{uv : u \in A \ \wedge v \in B\}$.

Teorema 5.2 Si A, B y C son lenguajes sobre Σ, se cumple que $(AB)C = A(BC)$.

Demostración: sea $AB = \{uv = w : u \in A \ \wedge v \in B\}$ y $(AB)C = \{wx : w \in AB \ \wedge x \in C\}$. Sustituyendo w, $(AB)C = \{(uv)x : uv \in AB \ \wedge x \in C\} = \{(uv)x : u \in A \ \wedge v \in B \wedge x \in C\}$. Ahora, de acuerdo al Teorema 5.1 $(AB)C = \{u(vx) : u \in A \ \wedge v \in B \wedge x \in C\} = \{u(vx) : u \in A \ \wedge vx \in BC\} = A(BC)$ ■

Teorema 5.3 Si A, B y C son lenguajes sobre Σ se cumple que $A(B \cup C) = AB \cup AC$ y $(B \cup C)A = BA \cup CA$ ■

Demostración. Sea $A(B \cup C) = \{uv : u \in A \ \wedge v \in (B \cup C)\}$. Si $v \in B$ entonces $uv \in AB$ en consecuencia $uv \in AB \cup BC$. Del mismo modo Si $v \in C$ entonces $uv \in AC$ por tanto $uv \in AB \cup BC$ en cualquiera de los casos, en conclusión $A(B \cup C) = AB \cup AC$. Con el mismo razonamiento se demuestra que $(B \cup C)A = BA \cup CA$.

[17] Los prefijos se denominan propios cuando la cadena es diferente al prefijo y de igual manera para los sufijos.

Definición 5.10 La **potencia de un lenguaje** L sobre Σ donde $n \in \mathbb{N}$ se define como L^n y se cumple que $L^0 = \{\lambda\}$ y $L^n = \underbrace{LLL \ldots L}_{n\ veces}$

Definición 5.11 La **estrella o clausura de Kleene** de un lenguaje L es la unión de todas las potencias de L y se denota por L^*.

$$L^* = \bigcup_{i \geq 0} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup L^{n+1} \cup \ldots$$

Definición 5.12 La **clausura positiva** de un lenguaje L es la unión de todas las potencias de L diferentes a L^0y se denota L^+.

$$L^+ = \bigcup_{i \geq 1} L^i = L^1 \cup L^2 \cup \ldots \cup L^n \cup L^{n+1} \cup \ldots$$

Teorema 5.4 $L^+ = L^*L = LL^*$.

Demostración: aplicando la Definición 5.11 y el Teorema 5.3 y se obtiene que $LL^* = L(L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots) = L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots = L^+$, de igual manera se demuestra que $L^+ = L^*L$ ■

Definición 5.13 Un **lenguaje regular** sobre Σ es el conjunto de todos los lenguajes que se forman a partir de los lenguajes básicos ∅, $\{\lambda\}$, $\{a\}$, para cada $a \in \Sigma$ por medio de las operaciones de unión y concatenación de lenguajes y estrella de Kleene.

Nota 5.2 Σ es un lenguaje regular sobre Σ.

Ejemplo 5.7 Sea $\Sigma = \{a, b\}$; el lenguaje regular sobre Σ de todas las cadenas que tienen exactamente una a unido con el conjunto que tiene sólo una b es el siguiente:

$$L = \{b\}^*\{a\}\{b\}^* \cup \{b\} = \{a, bab, bbabb, bbbabbb, \ldots\} \cup \{b\}$$

$$L = \{a, b, bab, bbabb, bbbabbb, \ldots\}.$$

Nota 5.3 La concatenación de lenguajes tiene prioridad sobre la operación unión.

Con el fin de representar la escritura de los lenguajes regulares de una forma más práctica y sencilla a continuación se define lo que es una expresión regular.

Definición 5.14 Las **expresiones regulares** sobre un alfabeto Σ dado, se definen como:

i. $\emptyset$ es una expresión regular que representa al lenguaje $\emptyset$.
ii. λ es una expresión regular que representa al lenguaje $\{\lambda\}$.
iii. a es una expresión regular que representa al lenguaje $\{a\}, a \in \Sigma$
iv. Si R y S son expresiones regulares sobre Σ, también lo son: RS, $R \cup S$ y R^*.

Ejemplo 5.8 Dado el alfabeto $\Sigma = \{a, b, c\}$ el siguiente es un lenguaje regular sobre Σ, $L = (\{a\} \cup \{b\}^*)\ \{a\}^*\{bc\}^*$. Una expresión regular para el lenguaje L es el siguiente: $(a \cup b^*)\ a^*(bc^*)$, en adelante $L = (a \cup b^*)\ a^*(bc^*)$.

6. Teoría de Autómatas

6.1 Autómatas

El estudio de los Autómatas finitos desde el punto de vista de las matemáticas se aborda en la Teoría Algebraica de Autómatas, allí se profundiza, formaliza y describe a partir del álgebra abstracta (comenzando con el reconocimiento de estructuras de semigrupo) un sin número de resultados con aplicaciones en la informática, la biología, psicología, entre muchas otras áreas del conocimiento humano (Holcombe, 1982, pp. ix-x). Desde el punto de vista algorítmico los autómatas permiten la manipulación de cadenas y de secuencias. Así pues, el estudio de los autómatas implica aproximarse a los modelos abstractos de dispositivos lógicos donde se analiza lo que se puede hacer y lo que no con ellos.

Los orígenes de la teoría de Autómatas se encuentran en la ingeniería eléctrica con Claude Elwood Shannon (1916-2001). Shannon demostró la aplicación álgebra booleana en el campo de los circuitos digitales en 1938 y luego se hizo famoso por su Teoría de la Información en 1948. Estas fueron las bases para la aplicación de la Lógica Matemática a los circuitos secuenciales (utilizando la idea de estado de un autómata y tablas de transición) dando lugar a la Teoría de las máquinas secuenciales y de los Autómatas finitos. Sin embargo, la biología hace parte también de la génesis con los estudios de McCulloch y Pitts (1943) donde describen cálculos lógicos para simular la actividad de una neurona, pues una red neuronal (natural o artificial) es una colección de procesadores elementales (neuronas), conectados entre sí, produciendo entradas y salidas (señales) (Navarrete et al., 2003, pp. 5-6).

La primera publicación formal sobre autómatas fue expuesta por el matemático S. C. Kleene (1909-1994) en 1954 donde expone el famoso “teorema de Kleene”. Este trabajo está basado en las investigaciones sobre el sistema nervioso de McCulloch y Pitts publicadas en 1943.

6.1.1 Autómatas finitos.

Un Autómata finito (AF)[18] es una construcción lógica que recibe a un conjunto de símbolos como entrada, luego produce una salida determinando si la cadena pertenece o no pertenece a determinado lenguaje (Ilustración 6.1):

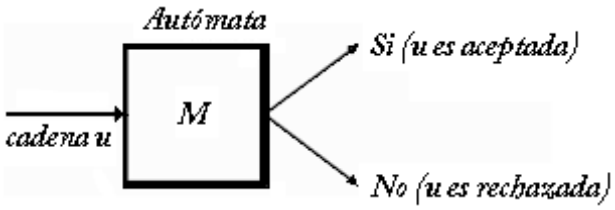

Ilustración 6.1 Autómata finito. Fuente (De Castro, 2004, p. 25)

Los AF son máquinas abstractas que tienen una unidad de control (UC)[19] que escanea o lee desde el extremo izquierdo de una cinta. Esta cinta contiene celdas sobre la cual se escribe una cadena de entrada. A cada símbolo de la cadena se le asigna una casilla en la cinta. Según sea la posición y el valor escaneado por la UC en la cinta, está tendrá asociada una función de transición δ la cual la moverá a la casilla derecha o inclusive puede permanecer quieta. Este proceso se realizará hasta que el autómata finalice con un estado de aceptación o de rechazo respecto a la cadena procesada (Ilustración 6.2).

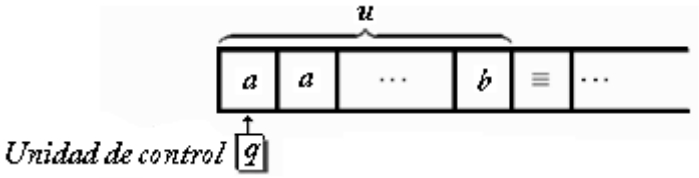

Ilustración 6.2 Esquema cinta, cadena y unidad de control. Fuente (De Castro, 2004, p. 26)

Definición 6.1 Un AF M está definido por cinco parámetros, $M = (\Sigma, Q, q_0, F, \delta)$:

i. Un alfabeto Σ. Todas las cadenas procesadas por un AF pertenecen a Σ^*.
ii. $Q = \{q_0, q_1, \dots, q_n\}$, conjunto finito de estados del autómata.
iii. $q_0 \in Q$, estado inicial.
iv. $F \subseteq Q$, conjunto de estados finales o de aceptación.

[18] En adelante se usará la abreviatura AF para Autómata finito.
[19] En adelante se usará la abreviatura UC para unidad de control.

v. La función de transición del autómata

$$\delta: Q \times \Sigma \to \mathrm{Q}$$
$$(q, a) \mapsto \delta(q, a)$$

Ejemplo 6.1 Sea M un AF definido por: $\Sigma = \{0,1\}$, $Q = \{q_0, q_1, q_2\}$, q_0: estado inicial, $F = \{q_0, q_1\}$ y la función de transición δ definida en la Tabla 6.1:

Tabla 6.1 Función de transición δ Ejemplo 6.1

Función de transición δ		
$\delta(q_0, 0) = q_0$	$\delta(q_1, 0) = q_1$	$\delta(q_2, 0) = q_1$
$\delta(q_0, 1) = q_1$	$\delta(q_1, 1) = q_2$	$\delta(q_2, 1) = q_1$

Otra manera de presentar la información de la Tabla 6.1 para la función de transición δ está dada por la Tabla 6.2. Para efectos prácticos en adelante se usará este último esquema.

Tabla 6.2 Función de transición δ Ejemplo 6.1

δ	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

A continuación se muestra el procesamiento de dos cadenas de entrada $u = 00101$ y $v = 001010$ donde se verificará si son o no aceptadas por M.

Para la cadena $u = 00101$ (véase Ilustración 6.3): desde el extremo izquierdo de la cinta están escritos los símbolos de la cadena u y la UC se encuentra bajo la primera celda en el estado inicial q_0.

La UC lee el primer símbolo de la cadena u, es decir 0 y debido a que $\delta(q_0, 0) = q_0$ esta se mueve hacia la derecha. Ahora la UC se encuentra bajo la segunda casilla nuevamente en el estado q_0. Por la razón anterior, la UC se mueve hacia la derecha y queda bajo la tercera casilla en el estado q_0. Ahora la UC en el estado q_0 lee el símbolo 1 y se mueve hacia la derecha quedando bajo la cuarta casilla en el estado q_1 debido a que $\delta(q_0, 1) = q_1$. Como $\delta(q_1, 0) = q_1$ la UC se mueve hacia la derecha y queda en el estado q_1. Por último como $\delta(q_1, 1) = q_2$ la UC se mueve hacia la derecha, queda en el estado q_2 y debido a que no hay símbolos bajo esta casilla el proceso termina.

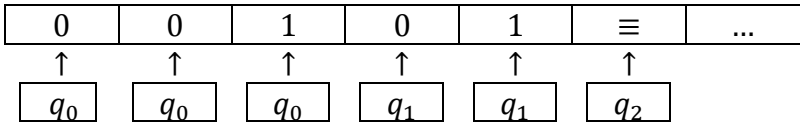

Ilustración 6.3 Procesamiento de la cadena *u* Ejemplo 6.1

Se debe colocar especial atención en último estado, pues si este pertenece al conjunto de los estados finales F la cadena es aceptada o de lo contrario es rechazada. Para este ejemplo $q_2 \notin F$ por lo tanto la cadena $u = 00101$ es rechazada.

Para la cadena $v = 001010$. La UC en el estado inicial q_0 lee el primer símbolo de la cadena v, es decir 0 y se mueve hacia la derecha quedando en el estado q_0. Puesto que $\delta(q_0, 0) = q_0$ la UC se mueve a la derecha y queda en el estado q_0 nuevamente. Como $\delta(q_0, 1) = q_1$ la UC se mueve a la derecha y queda en el estado q_1 y así sucesivamente. La Ilustración 6.4 expone de manera completa el análisis:

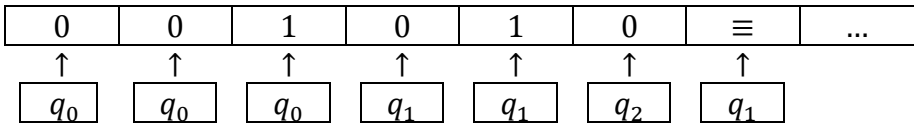

Ilustración 6.4 Procesamiento de la cadena *v* Ejemplo 6.1

Como $q_1 \in F$, entonces la cadena v es aceptada por el M. Obsérvese que $v = u0$ es una concatenación entre la cadena u y el símbolo 0.

Como conclusión, un AF al leer el último símbolo de una cadena determina un estado q_i y se puede presentar alguno de los siguientes casos:

i. Si $q_i \in F$, entonces la cadena será aceptada por el autómata.
ii. Si $q_i \notin F$, entonces la cadena será rechazada por el autómata.

Como un caso particular, la cadena vacía λ es aceptada por M pues inicialmente la UC se encuentra en el estado inicial, no se mueve, siendo este un estado de aceptación.

Un AF se puede representar mediante un grafo dirigido y etiquetado. La Ilustración 6.5 representa el autómata M con la información brindada por la Tabla 6.2. En este caso los nodos representan los estados q_0, q_1 y q_2; las funciones de transición están

representadas cada una con una flecha unidireccional de nodo a nodo con el respectivo símbolo que procesan, por ejemplo $\delta(q_1, 1) = q_2$ representa a aquella que va desde q_1 a q_2 y que sólo lee al símbolo 1; un triángulo apuntando hacia uno de los nodos, en este caso q_0, representa el estado inicial (sólo puede existir uno) y finalmente los estados finales q_0 y q_1 con un círculo interior en cada uno de los nodos:

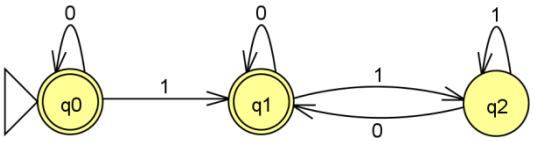

Ilustración 6.5 Grafo dirigido autómata *M* Ejemplo 6.1

Verificar si las cadenas $u = 00101$ y $v = 001010$ son aceptadas o rechazadas es un trabajo más sencillo por medio de un grafo dirigido, pues hay que determinar el camino por el que nos llevan los símbolos de la cadena a medida que son leídos por la UC pasando por cada uno de los estados en este caso representados por nodos[20].

6.1.2 Autómatas finitos deterministas y no deterministas.

Definición 6.2 Se denomina **autómata finito determinista** (AFD), a un AF que para cada estado q y símbolo $a \in \Sigma$ define una función de transición completa y unívoca.

Definición 6.3 Se denomina **autómata finito no-determinista** (AFN) a un autómata finito que para algún estado q y símbolo $a \in \Sigma$ existen por lo menos dos transiciones o estados para un mismo símbolo.

Nota 6.1 Un AFD M básicamente es una quíntupla definida por $M = (\Sigma, Q, q_0, F, \delta)$. De acuerdo a la Definición 6.3 un AFN M' es una quíntupla $M' = (\Sigma, Q, q_0, F, \Delta)$ donde la función de transición Δ no es uno a uno y se define por: $\Delta: Q \times \Sigma^* \rightarrow \wp(\mathrm{Q})$.

Ejemplo 6.2 A continuación se determinará si la cadena $u = 00011$ es aceptada por un AFN M, definido como $M = (\Sigma, Q, q_0, F, \Delta)$ del modo siguiente:

[20] Los grafos dirigidos de los Autómatas y las máquinas de Turing presentados a lo largo de este trabajo fueron diseñados y simulados en JFLAP; en el Anexo 1 se brinda la información básica acerca de esta aplicación. Por último y no menos importante, todas las simulaciones realizadas se encuentran disponibles de acuerdo al Anexo 2.

i. $\Sigma = \{0,1\}$.

ii. $Q = \{q_0, q_1, q_2, q_3\}$.

iii. q_0: estado inicial

iv. $F = \{q_1, q_3\}$.

v. Función de transición Δ (Tabla 6.3):

Tabla 6.3 Función de transición Δ Ejemplo 6.2

Δ	0	1
q_0	$\{q_0, q_1, q_3\}$	$\emptyset$
q_1	$\{q_1\}$	$\{q_2\}$
q_2	$\emptyset$	$\{q_1, q_2\}$
q_3	$\emptyset$	$\{q_3\}$

La Tabla 6.3 muestra casos de cómputo abortado, es decir la función de transición no está definida para algunos valores como $\Delta(q_0, 1) = \emptyset$, $\Delta(q_2, 0) = \emptyset$ y $\Delta(q_3, 0) = \emptyset$, mientras que para otros tiene diferentes estados asociados, $\Delta(q_0, 0)$ puede ser q_0, q_1 o q_3 determinando que M sea un AFN. La Ilustración 6.6 muestra el grafo dirigido de M:

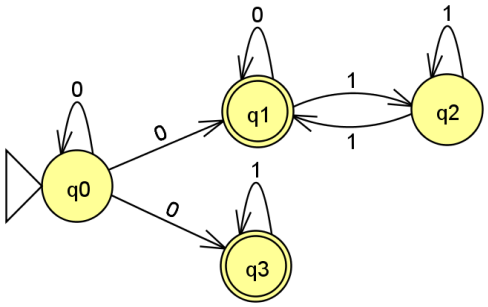

Ilustración 6.6 Grafo dirigido del AFN *M* Ejemplo 6.2

Respecto al procesamiento de la cadena $u = 00011$ por el AFN M, la UC se encuentra en el estado inicial q_0 y empieza por leer la cadena desde primer símbolo, en este caso 0, pero ¿en qué estado debe ubicarse la UC bajo la segunda casilla en la cinta? Hay varias alternativas puesto que $\Delta(q_0, 0) = \{q_0, q_1, q_3\}$. Se describen a continuación cuatro de ellas:

i. Primera alternativa (Ilustración 6.7): la UC se ubica bajo la segunda casilla en el estado q_3 y como no está definido $\Delta(q_3, 0)$, el cómputo automáticamente queda abortado.

0	0	0	1	1	≡	...
↑	↑					
q_0	q_3					

Ilustración 6.7 Primera alternativa procesamiento de la cadena *u* por el AFN *M* Ejemplo 6.2

ii. Segunda alternativa (Ilustración 6.8): la UC se ubica bajo la segunda casilla en el estado q_1. Debido a que $\Delta(q_1, 0) = \{q_1\}$ la UC se ubica bajo la tercera casilla en el estado q_1 nuevamente y por la misma razón salta a la siguiente casilla. Ahora la UC se ubica bajo la cuarta casilla en el estado q_1. Como $\Delta(q_1, 1) = \{q_2\}$, la UC se ubica bajo la quinta casilla en el estado q_2.

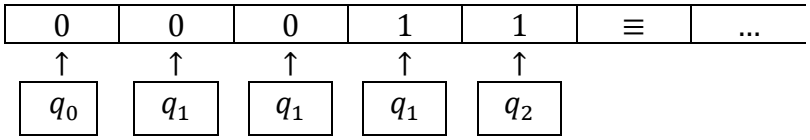

Ilustración 6.8 Segunda alternativa procesamiento de la cadena *u* por el AFN *M* Ejemplo 6.2

En este momento hay dos alternativas ya que $\Delta(q_2, 1) = \{q_1, q_2\}$. Se toma la primera q_1. Debido a que la cadena ya fue completamente leída y q_1 es un estado final, la cadena es aceptada. En cambio si se toma la segunda alternativa q_2 la cadena es rechazada puesto que $q_2 \notin F$. La Ilustración 6.9 representa el cómputo exitoso:

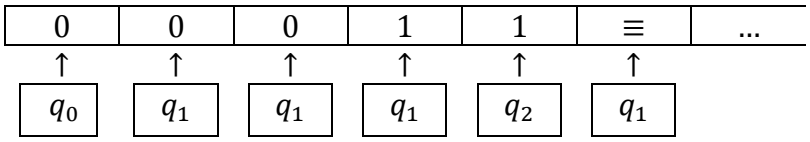

Ilustración 6.9 Procesamiento exitoso de la cadena *u* por el AFN *M* Ejemplo 6.2

iii. Tercera alternativa: la UC se ubica bajo la segunda casilla en el estado q_0 y pasa a la tercera casilla en el mismo estado también. Ahora la UC pasa a la cuarta casilla en el estado q_3 y finalmente pasa a la quinta casilla en este estado también. Este es un cómputo exitoso.

iv. Cuarta alternativa: la UC se ubica bajo la segunda casilla en el estado q_0. Repitiendo lo anterior y cuando la UC se encuentre en la tercera casilla estará en estado q_0 y como $\Delta(q_0, 1) = \emptyset$, el cómputo se aborta.

Una cadena es aceptada por un AFN si y sólo si existe un cómputo exitoso para ella, por lo tanto la cadena $u = 00011$ es aceptada por M. Ya conociendo cuando una cadena es aceptada por un AFN, se mencionará cuando se dice que un lenguaje es aceptado por un AFD y AFN.

Definición 6.4 Dado un autómata M, ya sea AFD o AFN, el lenguaje L **aceptado** por M se denota por $L(M)$ y se define como:

$$L(M) = \{u \in \Sigma^*: M\ procesa\ la\ cadena\ u\ y\ termina\ en\ un\ estado\ q \in F\}$$

Ejemplo 6.3 Conociendo que el lenguaje aceptado por el AFN M del Ejemplo 6.2 es el conjunto de todas las cadenas u que son aceptadas por M con al menos un cómputo exitoso, se analizará si para las siguientes cadenas u_i se cumple que $u \in L(M)$:

i. La cadena vacía $u_1 = \lambda$ es rechazada por M, por tanto $\lambda \notin L(M)$.
ii. $u_2 = 0 \ldots 0$ y en general cualquier cadena que contenga sólo ceros es aceptada por M. $u_2 \in L(M)$.
iii. La cadena que contiene cualquier cantidad de sólo unos $u_3 = 1 \ldots 1$ es rechazada por M, por tanto $u_3 \notin L(M)$. En general cualquier cadena que comienza con un uno sin importar el resto de símbolos de la cadena es rechazada.
iv. La cadena $u_4 = 01 \ldots 1$ que comienza con un cero y termina con cualquier cantidad de unos es aceptada por la máquina M $u_4 \in L(M)$.
v. La cadena $u_5 = 010$ es rechazada por M. $u_5 \notin L(M)$.
vi. La cadena $u_6 = 0 \ldots 010 \ldots 0$ es rechazada por M. $u_6 \notin L(M)$.
vii. La cadena $u_7 = 0110$ es aceptada por M. $u_7 \in L(M)$.
viii. La cadena que tenga un número par de unos entre dos ceros es aceptada.
ix. La cadena que tenga un número impar de unos entre dos ceros es rechazada.

Por último, se realiza el siguiente análisis para corroborar si los lenguajes L_i son aceptados por M:

i. $L_1 = 01^+0^*$, es un lenguaje que representa a todas las cadenas que comienzan por cero, seguido de cualquier cantidad de unos y finalmente por λ o cualquier cantidad de ceros. En general, este no es un lenguaje aceptado por M, pues la cadena $u = 010 \in L_1$ pero es rechazada por M, por tanto $L_1 \nsubseteq L(M)$.
ii. $L_2 = 01^+$, es un lenguaje que representa a todas las cadenas que comienzan por cero, seguido de cualquier cantidad de unos es un lenguaje aceptado por M. Se dice que $L_2 \subseteq L(M)$.

iii. $L_3 = 0^+$ es una lenguaje es un lenguaje aceptado por M. $L_3 \subseteq L(M)$.

Con el teorema de Kleene parte II, que se mostrará en la sección 6.2, se establecerá con exactitud el lenguaje que recibe un AF dado, en otras palabras, se determinará el conjunto de todas las cadenas posibles que son aceptadas por un AF.

6.1.3 Autómatas finitos no deterministas con transiciones lambda.

Definición 6.5 Un autómata finito no determinista con transiciones λ (AFN - λ), es un AFN**,** en el que la función de transición está definida como:

$$\Delta: Q \times (\Sigma \cup \{\lambda\}) \to \wp(Q)$$

Nota 6.2 Con el fin de precisar sobre la función de transición presentada en la Definición 6.5, la función $\Delta(q, \lambda) = \{q, q_{i_1}, \dots, q_{i_n}\}$ y que se denomina de transición λ o transición nula, tiene el fin de cambiar o no el estado sin mover de posición la UC sin leer o procesar el símbolo sobre la cinta y sólo en los casos para los cuales esta función transición que esté definida.

Ejemplo 6.4 Se determinará si las siguientes cadenas $u = 001$ y $v = 0100$ son aceptadas por el siguiente AFN - λ M definido como:

i. $\Sigma = \{0,1\}$.
ii. $Q = \{q_0, q_1, q_2, q_3, q_4\}$.
iii. q_0: estado inicial.
iv. $F = \{q_2, q_4\}$.
v. Función de transición Δ (Tabla 6.4):

Tabla 6.4 Función de transición Δ Ejemplo 6.4

Δ	0	1	λ
q_0	$\{q_3\}$	$\emptyset$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\{q_2\}$	$\emptyset$
q_3	$\emptyset$	$\{q_3\}$	$\{q_4\}$
q_4	$\{q_4\}$	$\emptyset$	$\emptyset$

El diágrafo de la máquina M se muestra en la Ilustración 6.10:

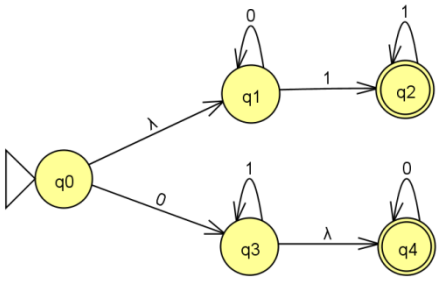

Ilustración 6.10 Diágrafo AFN- λ *M* Ejemplo 6.4

Procesando $u = 001$: la UC en el estado q_0 lee el primer símbolo de la cadena. Se presentan dos posibilidades, pero sólo se analizará la que llevará a la aceptación de la cadena. La UC hace una transición lambda y cambia al estado q_1 sin cambiar de casilla. Esto se puede hacer ya que la transición lambda está perfectamente definida de q_0 a q_1. Luego de varias transiciones la cadena es aceptada de acuerdo a la Ilustración 6.11:

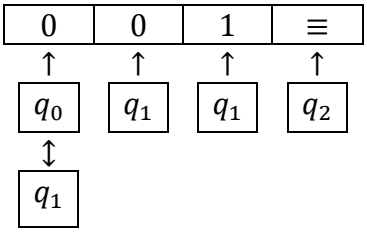

Ilustración 6.11 Procesamiento exitoso cadena *u* Ejemplo 6.4

Procesando $v = 0100$ se obtiene exitosamente (Ilustración 6.12):

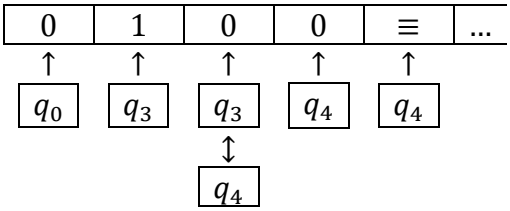

Ilustración 6.12 Procesamiento exitoso cadena *v* Ejemplo 6.4

Si se computa convenientemente la cadena v es aceptada por el AFN – λ M. Obsérvese que en el tercer símbolo la UC cambia del estado q_3 a q_4 automáticamente por medio de la transición lambda previamente definida**.**

Nota 6.3 En el ejemplo 6.4, es importante tener en cuenta que aunque la cadena u se pueda reescribir como $u = \lambda 001$ (esta igualdad es correcta) la UC en el estado inicial jamás procesará λ para pasar a otro estado (en esto no consiste la transición lambda), pues λ a pesar de ser un símbolo precisamente simboliza la carencia de estos, como se mencionó anteriormente es la cadena vacía.

6.1.4 Autómatas finitos con pila deterministas.

Definición 6.6 Se denomina **autómata con pila determinista** (AFPD) a una 7–tupla $M = (Q, q_0, F, \Sigma, \Gamma, \mathrm{Z}, \Delta)$**,** con los siguientes componentes:

i. Q es un conjunto finito de estados.
ii. $q_0 \in Q$ es el estado inicial.
iii. F es el conjunto de estados finales o de aceptación, $\emptyset \neq F \subseteq Q$.
iv. Σ alfabeto de la cinta.
v. Γ es el alfabeto de la pila.
vi. $\mathrm{Z} \in \Gamma$ es el símbolo inicial de la pila.
vii. Δ es la función de transición del autómata: $\Delta: Q \times (\Sigma \cup \lambda) \times \Gamma \to (Q \times \Gamma^*)$

A diferencia de los autómatas hasta ahora presentados, los AFPD utilizan dos cintas, donde una es la de entrada (horizontal) y la segunda de almacenamiento (denominada de pila) y que se coloca verticalmente (ver Ilustración 6.13). La UC escanea un símbolo sobre la cinta de entrada a y al mismo tiempo el símbolo Z en el tope de la pila.

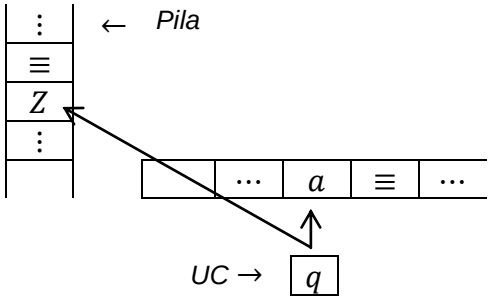

Ilustración 6.13 Esquema general AFPD Definición 6.6

La transición $\Delta(q, a, Z) = (q', \gamma)$, se representa en la Ilustración 6.14; la UC siempre se mueve a la derecha; $\gamma = a_0 \ldots a_n \in \Gamma^*$ es una cadena de caracteres que se escribe de arriba hacia abajo remplazando a Z. Si se ignora la pila del AFPD M se puede ver como un AFD**.** Las siguientes son transiciones especiales de un AFPD:

i. $\Delta(q, a, \mathrm{Z}) = (q', \mathrm{Z})$ el contenido de la pila no se altera, sin embargo hay un cambio de estado y la UC se mueve a la derecha.
ii. $\Delta(q, a, \mathrm{Z}) = (q', \lambda)$ el símbolo en el tope de la pila, que es Z, se borra y se pasa a leer el nuevo tope de la pila o primer símbolo de λ. Hay un cambio de estado y la UC se mueve a la derecha.
iii. $\Delta(q, \lambda, s) = (q', \gamma)$ está se denomina transición espontanea, la UC no lee el símbolo sobre la cinta y por lo tanto no se mueve el cabezal, pero el símbolo del tope de la pila es reemplazado por la cadena γ. Esta es una transición lambda del AFPD. Hay un cambio de estado.

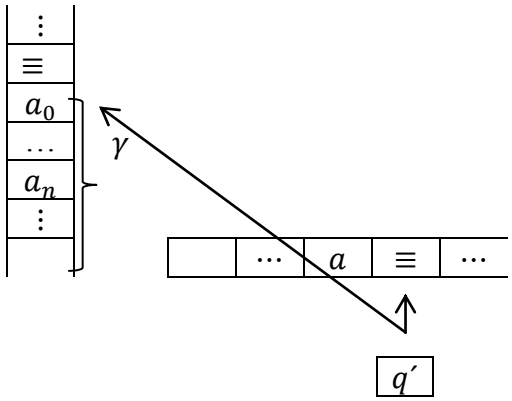

Ilustración 6.14 Transición $\Delta(q,a,Z)=(q',\gamma)$ Definición 6.6

Una cadena o lenguaje es aceptada si al procesar la cadena de entrada, ésta termina en un estado de aceptación sin importar la cadena compilada en la pila.

Describir todas las transiciones Δ de una AFPD es una tarea muy engorrosa, pues se tendrían tablas de tripe entrada. Una manera cómoda para representar una transición cualquiera $\Delta(q, a, Z) = (p, \gamma)$ es la siguiente:

$$(q, au, Z\beta) \vdash (p, u, \gamma\beta)$$

Esta notación nos dice que la UC se encuentra bajo el símbolo a en el estado q, u es un sufijo de la cadena au y la cadena $Z\beta$ es el contenido de la pila donde Z está en el tope. Luego de la transición la UC pasa a estar debajo del primer símbolo de la cadena u y el tope de la pila Z es remplazado por la cadena γ. Ahora la notación:

$$(q, u, \beta) \vdash^* (p, v, \gamma)$$

Establece que en $i \geq 0$ pasos se pasa de la transición Δ_1 a Δ_i.

Definición 6.7 Un lenguaje L aceptado por una AFPD M se define como:

$$L(M) = \{w \in \Sigma^* : (q_0, w, \mathrm{Z}) \vdash^* (q', \lambda, \beta), q' \in F\}$$

Una cadena es aceptada si en cero, uno o más pasos se llega a un estado final.

Nota 6.4 No existen cómputos con pila vacía, es decir se requiere por lo menos un símbolo en la pila al realizar una función de transición.

Ejemplo 6.5 El siguiente AFPD M acepta el lenguaje $L = \{0^i 1^{2i} / i \geq 1\}$ cuyas cadenas tienen una cantidad i de ceros y $2i$ de unos seguidos con $i \geq 1$.

i. $Q = \{q_0, q_1, q_2, q_3\}$
ii. $q_0 \in Q$ es el estado inicial.
iii. $F = \{q_3\}$
iv. $\Sigma = \{0,1\}$
v. $\Gamma = \{Z, C\}$
vi. $\mathrm{Z} \in \Gamma$ símbolo inicial de la pila.
vii. La función de transición Δ está dada por la Tabla 6.5:

Tabla 6.5 Función de transición Δ AFPD M

$\Delta_0(q_0, 0, Z) = (q_1, CCZ)$
$\Delta_1(q_1, 0, Z) = (q_1, CC)$
$\Delta_2(q_1, 0, C) = (q_1, CCC)$
$\Delta_3(q_1, 1, C) = (q_2, \lambda)$
$\Delta_4(q_2, 1, C) = (q_2, \lambda)$
$\Delta_5(q_2, \lambda, Z) = (q_3, Z)$

A continuación se procesarán las cadenas $u = 001111$ y $v = 001011111$; como $u \in L$ el AFPD M deberá aceptar la cadena, mientras que v debe ser rechazada por el autómata dado que $v \notin L$.

Cadena u: el proceso de la cadena inicia con $(q_0, 001111, Z)$, usando la transición Δ_0 tenemos:

$$(q_0, 001111, Z) \vdash (q_1, 01111, CCZ)$$

La primera transición hace un cambio de estado de q_0 a q_1, el tope en la pila Z es remplazado por CCZ, el cabezal se mueve hacia la derecha para leer el siguiente símbolo de la cadena que es 0. Luego se usa la transición Δ_2, para obtener:

$$(q_1, 01111, CCZ) \vdash (q_1, 1111, CCCCZ)$$

Donde el tope de la pila que es C es remplazado por CCC. Ahora se utiliza la transición Δ_3:

$$(q_1, 1111, CCCCZ) \vdash (q_2, 111, CCCZ)$$

El tope de la pila C es remplazado por λ, en otras palabras el símbolo del tope de la pila es borrado. Con Δ_4 se borrar el contenido de la pila y se tiene:

$$(q_2, 111, CCCZ) \vdash (q_2, 11, CCZ) \vdash (q_2, 1, CZ) \vdash (q_2, \lambda, Z)$$

Por último usamos la transición Δ_5 y se llega a un estado de aceptación para u.

$$(q_2, \lambda, Z) \vdash (q_3, \lambda, Z)$$

Cadena v: se inicia con la transición Δ_0:

$$(q_0, 001011111, Z) \vdash (q_1, 01011111, CCZ)$$

Luego se usa la transición Δ_2 y se obtiene:

$$(q_1, 01011111, CCZ) \vdash (q_1, 1011111, CCCCZ)$$

Con la transición Δ_3 se borrará el contenido del tope de la pila:

$$(q_1, 1011111, CCCCZ) \vdash (q_2, 011111, CCCZ)$$

Como no existe una transición definida para $(q_2, 011111, CCCZ)$ entonces la cadena v es rechazada o abortada por el AFPD M.

El diágrafo correspondiente a este AFPD M corresponde a la Ilustración 6.15. Sobre las flechas se indican las transiciones de la máquina, en particular del nodo de q_0 a q_1 se encuentra $0, Z; CCZ$ que corresponde a la transición $\Delta_0(q_0, 0, Z) = (q_1, CCZ)$, la cual

lee al 0 si Z está en el tope de la pila para posteriormente ser remplazado por la cadena CCZ.

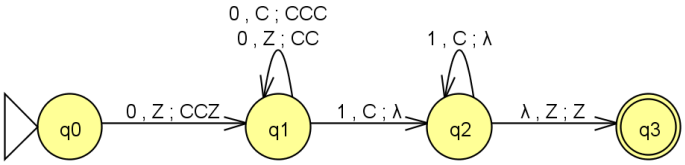

Ilustración 6.15 AFPD *M* Ejemplo 6.5

6.1.5 Autómatas finitos con pila no deterministas.

Definición 6.8 Se denomina autómata finito con pila no-determinista (AFPN), a una 7–tupla $M = (Q, q_0, F, \Sigma, \Gamma, \mathrm{Z}, \Delta)$, con mismos componentes de un AFPD pero con la función de transición se definida así:

$$\Delta: Q \times (\Sigma \cup \lambda) \times \Gamma \to \wp(Q \times \Gamma^*)$$

Ejemplo 6.6 El siguiente AFPN M acepta el lenguaje $L = \{a^i b^j c^{i+j} / i, j \geq 0\}$.

i. $Q = \{q_0, q_1, q_2\}$
ii. $q_0 \in Q$ es el estado inicial.
iii. $F = \{q_2\}$
iv. $\Sigma = \{a, b, c\}$
v. $\Gamma = \{\mathrm{Z}, A, B\}$
vi. $\mathrm{Z} \in \Gamma$ símbolo inicial de la pila.
vii. La función de transición Δ definida en la Tabla 6.6:

Tabla 6.6 Función de transición Δ del AFPN Ejemplo 6.6

$\Delta_0(q_0, a, Z) = (q_0, AZ)$	$\Delta_4(q_0, b, B) = (q_0, BB)$	$\Delta_8(q_1, c, B) = (q_1, \lambda)$
$\Delta_1(q_0, b, Z) = (q_0, BZ)$	$\Delta_5(q_0, c, A) = (q_1, \lambda)$	$\Delta_9(q_1, \lambda, Z) = (q_2, Z)$
$\Delta_2(q_0, a, A) = (q_0, AA)$	$\Delta_6(q_0, c, B) = (q_1, \lambda)$	$\Delta_{10}(q_0, \lambda, Z) = (q_2, Z)$
$\Delta_3(q_0, b, A) = \{(q_0, BA), (q_0, AB)\}$	$\Delta_7(q_1, c, A) = (q_1, \lambda)$	

Se procesarán dos cadenas $u = aabccc$ y $v = abccac$. Puesto que $u \in L$ entonces el autómata M deberá aceptar la cadena, mientras que v debe ser rechazada por el autómata dado que $v \notin L$. Procesando u:

$$(q_0, aabccc, Z) \vdash (q_0, abccc, AZ) \vdash (q_0, bccc, AAZ)$$

Ahora con la transición Δ_3 existen dos posibilidades. Se utilizará la segunda opción:

$$(q_0, bccc, AAZ) \vdash (q_0, ccc, ABAZ).$$

Finalmente para llegar a un cómputo exitoso se necesitan los siguientes pasos computacionales:

$$(q_0, ccc, ABAZ) \vdash (q_1, cc, BAZ) \vdash (q_1, c, AZ) \vdash (q_1, \lambda, Z) \vdash (q_2, \lambda, Z)$$

Procesando v:

$$(q_0, abccac, Z) \vdash (q_0, bccac, AZ)$$

Con la transición Δ_3 se pueden escoger una de dos, usando la primera opción, se obtiene:

$$(q_0, bccac, AZ) \vdash (q_0, ccac, BAZ)$$

Continuando:

$$(q_0, ccac, BAZ) \vdash (q_1, cac, AZ) \vdash (q_1, ac, Z)$$

Por este camino la función de transición no está definida para $\Delta(q_1, a, Z)$, por lo tanto la cadena u de momento es rechazada por el AFPD M. Se confirmará si u es rechazada por completo revisando el siguiente camino:

$$(q_0, abccac, Z) \vdash (q_0, bccac, AZ) \vdash (q_0, ccac, ABZ) \vdash (q_0, cac, BZ) \vdash (q_1, ac, Z)$$

Y como $\Delta(q_1, ac, Z)$ no está definida, la cadena u definitivamente es rechazada. El diágrafo correspondiente al AFPN *M* se muestra en la Ilustración 6.16:

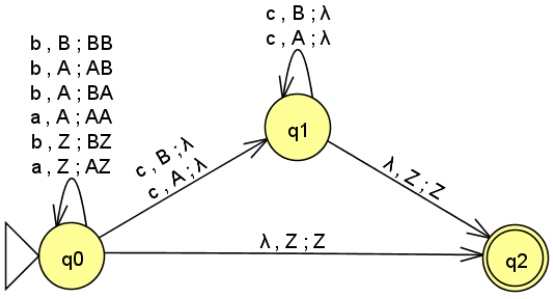

Ilustración 6.16 Diágrafo AFPN *M* Ejemplo 6.6

6.2 Teoremas sobre Autómatas

El primer teorema propuesto en esta sección establece que se puede construir un AFD que acepte las mismas cadenas o lenguaje que un AFN si se elimina el no determinismo del AFN. Como se ha mencionado, la diferencia esencial entre los AFN y los AFD se encuentra en la función de transición, por lo que es necesario extender las definiciones de las funciones δ y Δ.

Definición 6.9 Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD**.** La función de transición δ se extiende a $\bar{\delta}: Q \times \Sigma^* \to Q$ definida recursivamente por:

i. $\bar{\delta}(q, \lambda) = q, q \in Q$
ii. $\bar{\delta}(q, a) = \delta(q, a), q \in Q, a \in \Sigma$
iii. $\bar{\delta}(q, wa) = \delta\big(\bar{\delta}(q, w), a\big), q \in Q, a \in \Sigma, w \in \Sigma^*$

Definición 6.10 Para $a \in \Sigma$, $S \in \wp(Q)$ se define $\Delta(S, a) = \bigcup_{q \in S} \Delta(q, a)$.

Definición 6.11 Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. La función de transición Δ se extiende a $\bar{\Delta}: Q \times \Sigma^* \to \wp(Q)$, definida recursivamente por:

i. $\bar{\Delta}(q, \lambda) = \{q\}, q \in Q$
ii. $\bar{\Delta}(q, a) = \Delta(q, a), q \in Q, a \in \Sigma$
iii. $\bar{\Delta}(q, wa) = \Delta(\bar{\Delta}(q, w), a) = \bigcup_{p \in \bar{\Delta}(q,w)} \Delta(p, a)\ , q \in Q, a \in \Sigma, w \in \Sigma^*$

Teorema 6.1 (Equivalencia computacional AFN ≡ AFD) Dado un AFD existe un AFN equivalente y viceversa, dado un AFN existe un AFD equivalente.

Demostración parte i (Dado un AFD existe un AFN equivalente): los AFD son un caso particular de los AFN ya que los estados de los AFD se pueden visualizar como subconjuntos con un solo estado como elemento. Ahora se demostrará el recíproco:

Reformulación[21]: dado un AFN $M = \{\Sigma, Q, q_0, F, \Delta\}$ se puede construir un AFD M' equivalente a M, tal que $L(M) = L(M')$.

[21] Dos máquinas son equivalentes si estas aceptan exactamente los mismos lenguajes.

Demostración parte ii (dado un AFN existe un AFD equivalente): dado el AFN $M = \{\Sigma, Q, q_0, F, \Delta\}$ se construye un AFD $M' = \{\Sigma, \wp(Q), \{q_0\}, F', \delta\}$, con $F' = \{T \subseteq \wp(Q) : T \ contiene \ al \ menos \ un \ estado \ de \ aceptación \ de \ F\}$ y δ definido por:

$$\delta: \wp(Q) \times \Sigma \to \wp(Q)$$
$$(S, a) \mapsto \delta(S, a) = \Delta(S, a)$$

Por inducción matemática se demostrará que $L(M) = L(M')$ sobre una cadena $w \in \Sigma^*$ mostrando que $\delta(\{q_0\}, w) = \Delta(q_0, w)$.

$w = \lambda$:

$\delta(\{q_0\}, \lambda) = \Delta(\{q_0\}, \lambda)$ Definición de la función δ pues $\delta(S, a) = \Delta(S, a)$.
$= \bigcup_{q \in \{q_0\}} \Delta(q, \lambda)$ Definición 6.10.
$= \Delta(q_0, \lambda)$ Definición de unión de colecciones.
$= \{q_0\}$ Definición 6.11. parte i y ii.

$w = a$, $a \in \Sigma$:

$\delta(\{q_0\}, a) = \Delta(\{q_0\}, a)$ Definición de la función δ pues $\delta(S, a) = \Delta(S, a)$.
$= \bigcup_{q \in \{q_0\}} \Delta(q, a)$ Definición 6.10.
$= \Delta(q_0, a)$ Definición de unión de colecciones.

Haciendo uso de las funciones de extensión de las definiciones 6.9 y 6.11, puesto que permiten trabajar con cadenas, se propone como hipótesis de inducción que $\bar{\delta}(\{q_0\}, w) = \bar{\Delta}(\{q_0\}, w) = \bar{\Delta}(q_0, w)$ y que $a \in \Sigma$.

$\bar{\delta}(\{q_0\}, wa) = \delta\left(\bar{\delta}(\{q_0\}, w), a\right)$ Definición 6.9. parte iii.
$= \delta(\bar{\Delta}(\{q_0\}, w), a)$ Hipótesis de inducción.
$= \Delta(\bar{\Delta}(\{q_0\}, w), a)$ Definición de la función δ pues $\delta(S, a) = \Delta(S, a)$.
$= \Delta(\bar{\Delta}(q_0, w), a)$ Hipótesis de inducción.
$= \bar{\Delta}(q_0, wa)$ Definición 6.9. parte iii ■

Nota 6.5 A pesar de que la demostración del Teorema 6.1 (segunda parte) provee una manera de construir un AFD a partir de un AFN, este proceso puede ser muy engorroso e ineficiente si no se hace una restricción sobre conjunto S. En principio se

deben realizar todas las transiciones $\delta(S,a)$ para todo conjunto $S \in \wp(Q)$. Por tanto se restringen las funciones de transición δ sólo para los conjuntos donde S tiene un solo elemento y para aquellos que se van generando a partir de los singleton anteriores. El Ejemplo 6.6 mostrará en detalle el procedimiento y la restricción en mención.

Ejemplo 6.7 Se diseñará un AFD M' equivalente al siguiente AFN definido como $M = (\Sigma, Q, q_0, F, \Delta)$ (Ilustración 6.17) con:

i. $\Sigma = \{0,1\}$.
ii. $Q = \{q_0, q_1, q_2, q_3\}$.
iii. q_0: estado inicial
iv. $F = \{q_3\}$.
v. Función de transición Δ (Tabla 6.7):

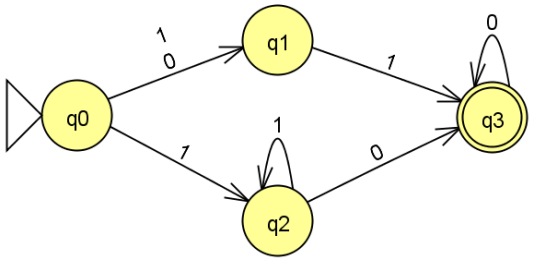

Ilustración 6.17 Diágrafo AFN M Ejemplo 6.7

Tabla 6.7 Función de transición Δ Ejemplo 6.7

Δ	0	1
q_0	$\{q_1\}$	$\{q_1, q_2\}$
q_1	$\emptyset$	$\{q_3\}$
q_2	$\{q_3\}$	$\{q_2\}$
q_3	$\{q_3\}$	$\emptyset$

Solución: Utilizando la demostración del Teorema 6.1 (segunda parte) se construye el AFD M' pedido utilizando la función de transición $(S,a) \mapsto \delta(S,a) = \Delta(S,a)$, donde $S \in \wp(Q)$ y $\Delta(S,a)$ está dado por la Definición 6.10.

$(\{q_0\},0) \mapsto \delta(\{q_0\},0) = \Delta(\{q_0\},0) = \bigcup_{q\in\{q_0\}} \Delta(q,0) = \Delta(q_0,0) = \{q_1\}.$

$(\{q_0\},1) \mapsto \delta(\{q_0\},1) = \Delta(\{q_0\},1) = \bigcup_{q\in\{q_0\}} \Delta(q,1) = \Delta(q_0,1) = \{q_1,q_2\}.$

$(\{q_1\},0) \mapsto \delta(\{q_1\},0) = \Delta(\{q_1\},0) = \bigcup_{q\in\{q_1\}} \Delta(q,0) = \Delta(q_1,0) = \emptyset.$

$(\{q_1\},1) \mapsto \delta(\{q_1\},1) = \Delta(\{q_1\},1) = \bigcup_{q\in\{q_1\}} \Delta(q,1) = \Delta(q_1,1) = \{q_3\}.$

$(\{q_2\},0) \mapsto \delta(\{q_2\},0) = \Delta(\{q_2\},0) = \bigcup_{q\in\{q_2\}} \Delta(q,0) = \Delta(q_2,0) = \{q_3\}.$

$(\{q_2\},1) \mapsto \delta(\{q_2\},1) = \Delta(\{q_2\},1) = \bigcup_{q\in\{q_2\}} \Delta(q,1) = \Delta(q_2,1) = \{q_2\}.$

$(\{q_3\},0) \mapsto \delta(\{q_3\},0) = \Delta(\{q_3\},0) = \bigcup_{q\in\{q_3\}} \Delta(q,0) = \Delta(q_3,0) = \{q_3\}.$

$(\{q_3\},1) \mapsto \delta(\{q_3\},1) = \Delta(\{q_3\},1) = \bigcup_{q\in\{q_3\}} \Delta(q,1) = \Delta(q_3,1) = \emptyset.$

Las transiciones anteriores se realizan solamente con conjuntos S unitarios. Uno de los resultados de estas transiciones es $\{q_1, q_2\} \in \wp(Q)$, por tanto atendiendo lo mencionado en la Nota 6.5 se procede del modo siguiente:

$$(\{q_1, q_2\}, 0) \mapsto \delta(\{q_1, q_2\}, 0) = \Delta(\{q_1, q_2\}, 0) = \bigcup_{q \in \{q_1, q_2\}} \Delta(q, 0) = \Delta(q_1, 0) \cup \Delta(q_2, 0).$$
$$= \emptyset \cup \{q_3\} = \{q_3\}$$
$$(\{q_1, q_2\}, 1) \mapsto \delta(\{q_1, q_2\}, 1) = \Delta(\{q_1, q_2\}, 1) = \bigcup_{q \in \{q_1, q_2\}} \Delta(q, 1) = \Delta(q_1, 1) \cup \Delta(q_2, 1).$$
$$= \{q_3\} \cup \{q_2\} = \{q_2, q_3\}$$

Uno de los resultados de las dos transiciones anteriores es $\{q_2, q_3\} \in \wp(Q)$, por tanto:

$$(\{q_2, q_3\}, 0) \mapsto \delta(\{q_2, q_3\}, 0) = \Delta(\{q_2, q_3\}, 0) = \bigcup_{q \in \{q_2, q_3\}} \Delta(q, 0) = \Delta(q_2, 0) \cup \Delta(q_3, 0).$$
$$= \{q_3\} \cup \{q_3\} = \{q_3\}$$
$$(\{q_2, q_3\}, 1) \mapsto \delta(\{q_2, q_3\}, 1) = \Delta(\{q_2, q_3\}, 1) = \bigcup_{q \in \{q_1, q_2\}} \Delta(q, 1) = \Delta(q_2, 1) \cup \Delta(q_3, 1).$$
$$= \{q_2\} \cup \emptyset = \{q_2\}$$

La Tabla 6.8 muestra la nueva de transiciones δ y la Ilustración 6.18 el diágrafo correspondiente:

Tabla 6.8 Función δ del nuevo AFD Ejemplo 6.7

δ	0	1
$\{q_0\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_1\}$	$\emptyset$	$\{q_3\}$
$\{q_2\}$	$\{q_3\}$	$\{q_2\}$
$\{q_3\}$	$\{q_3\}$	$\emptyset$
$\{q_1, q_2\}$	$\{q_3\}$	$\{q_2, q_3\}$
$\{q_2, q_3\}$	$\{q_3\}$	$\{q_2\}$

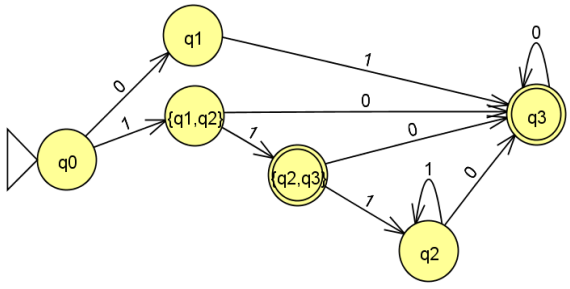

Ilustración 6.18 Diágrafo nuevo AFD Ejemplo 6.7

Cualquier otro subconjunto $S \in \wp(Q)$ diferente de los anteriores (primera columna Tabla 6.8) agrega nodos inservibles para la nueva máquina AFD, por ejemplo:

$$(\{q_0, q_3\}, 0) \mapsto \delta(\{q_0, q_3\}, 0) = \Delta(\{q_0, q_3\}, 0) = \bigcup_{q \in \{q_0, q_3\}} \Delta(q, 0) = \Delta(q_0, 0) \cup \Delta(q_3, 0).$$
$$= \{q_1\} \cup \{q_3\} = \{q_1, q_3\}$$

$$(\{q_0, q_3\}, 1) \mapsto \delta(\{q_0, q_3\}, 1) = \Delta(\{q_0, q_3\}, 1) = \cup_{q \in \{q_0, q_3\}} \Delta(q, 1) = \Delta(q_0, 1) \cup \Delta(q_3, 1).$$
$$= \{, q_1, q_2\} \cup \emptyset = \{, q_1, q_2\}$$

Se obtendrá un diágrafo como se muestra en la Ilustración 6.19 con dos nodos inutilizables pues el estado inicial es solamente q_0:

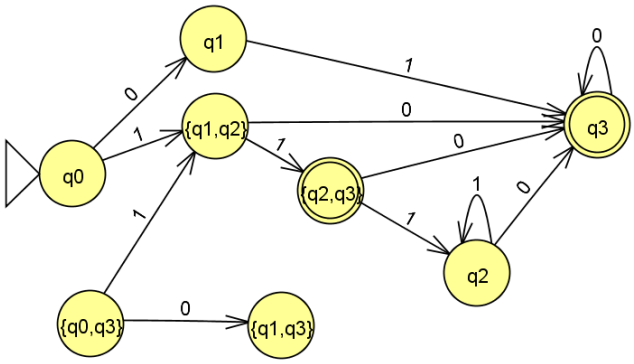

Ilustración 6.19 Diágrafo con nodos inutilizables Ejemplo 6.7

También existe una equivalencia entre los AFN – λ y los AFN. Esta equivalencia se logra si se eliminan las transiciones λ y además añadiendo transiciones que las simulen sin alterar el lenguaje que aceptan ambas máquinas aceptan. Para mostrar esta equivalencia se utilizará la definición de lambda clausura.

Definición 6.12 Para un estado $\{q\} \in \wp(Q)$, la $\lambda - clausura$ de q, notada por $\lambda[\{q\}]$ es el conjunto de estados de M a los que se puede llegar desde q por cero, una o más transiciones lambda.

Ejemplo 6.8 Dado el AFN – λ M de la Ilustración 6.20 se calculará $\lambda[\{q\}]$ para cada uno de los estados del autómata (ver Tabla 6.9):

Tabla 6.9 λ clausura de *M* Ejemplo 6.8

$\lambda[\{q_0\}] = \{q_0, q_1, q_3\}$
$\lambda[\{q_1\}] = \{q_1, q_3\}$
$\lambda[\{q_2\}] = \{q_2, q_3\}$
$\lambda[\{q_3\}] = \{q_3\}$

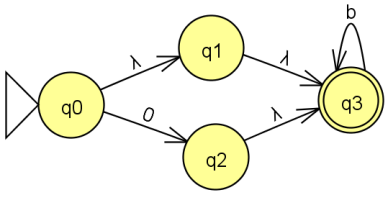

Ilustración 6.20 AFN – λ Ejemplo 6.8

Nota 6.6 No debe confundirse $\lambda[\{q\}]$ con $\Delta(q,\lambda)$, además $q \in \lambda[\{q\}]$ o también $\{q\} \subseteq \lambda[\{q\}]$.

Nota 6.7 Se puede extender la definición de $\lambda - clausura$ para un conjunto de estados. Por ejemplo para $\{q_1, \dots, q_k\}$ se tiene que $\lambda[\{q_1, \dots, q_k\}] = \lambda[\{q_1\}] \cup \dots \cup \lambda[\{q_k\}]$ donde $\lambda[\emptyset] = \emptyset$.

Teorema 6.2 (Equivalencia computacional AFN – $\lambda \equiv$ AFN) Dado un AFN existe un AFN – λ equivalente, y viceversa, Dado un AFN – λ existe un AFN equivalente.

Demostración parte i (Dado un AFN existe un AFN – λ equivalente): un AFN $M = \{\Sigma, Q, q_0, F, \Delta\}$ es un AFN – λ con cero transiciones lambda.

Reformulación: Dado un AFN – λ $M = \{\Sigma, Q, q_0, F, \Delta\}$ se puede construir un AFN M' equivalente a M, tal que $L(M) = L(M')$.

Demostración parte ii (Dado un AFN – λ existe un AFN equivalente): Se construye un AFN $M' = \{\Sigma, Q, q_0, F', \Delta'\}$ con Δ' definido por:

$$\Delta' : Q \times \Sigma \to \wp(Q)$$
$$(q, a) \mapsto \Delta'(q, a) = \lambda[\Delta(\lambda[\{q\}], a)]$$

y $F' = \{q \in Q : \lambda[q]\ contiene\ al\ menos\ un\ estado\ de\ aceptación\ de\ F\}$.

Por inducción matemática se demostrará que $L(M) = L(M')$ sobre una cadena $w \in \Sigma^*$ mostrando que $\Delta'(q_0, w) \cap \Delta(q_0, w) \neq \emptyset$ siempre y cuando $\Delta'(q_0, w)$ y $\Delta(q_0, w)$ sean ambos no vacíos.

$w = \lambda$:

$\Delta'(q_0, \lambda) = \{q_0\}$ por la Definición 6.11 parte ii y luego parte i ya que Δ' es una función de transición de un AFN. $\{q_0\} \subseteq \Delta(q_0, \lambda)$ por la Definición 6.5 y la Nota 6.2 ya que Δ es una función de transición de un AFN – λ. En conclusión $\Delta'(q_0, \lambda) \cap \Delta(q_0, \lambda) = \{q_0\}$.

$w = a$, $a \in \Sigma$:

$\Delta'(q_0, a) = \lambda[\Delta(\lambda[\{q_0\}], a)]$ por la definición de la función Δ'. Ahora $q_0 \in \lambda[\{q_0\}]$ por la Nota 6.6. Finalmente $\Delta(q_0, a) \subseteq \lambda[\Delta(\lambda[\{q_0\}], a)] = \Delta'(q_0, a)$, por lo que $\Delta'(q_0, a) \cap \Delta(q_0, a) \neq \emptyset$

Hipótesis de inducción: supóngase $\Delta'(q_0, w) \cap \Delta(q_0, w) \neq \emptyset$ y que $a \in \Sigma$.

$\Delta'(q_0, wa) = \Delta(\Delta'(q_0, w), a)$ por la Definición 6.11 parte iii. Como $\Delta'(q_0, w)$ y $\Delta(q_0, w)$ son no vacíos y $\Delta'(q_0, w) \cap \Delta(q_0, w) \neq \emptyset$ (hipótesis de inducción), se tiene que $\Delta(\Delta'(q_0, w) \cap \Delta(q_0, w), a) \subseteq \Delta(\Delta'(q_0, w), a) = \Delta'(q_0, wa)$.

$\Delta(q_0, wa) = \Delta(\Delta(q_0, w), a)$ por la Definición 6.11 parte iii. Como $\Delta'(q_0, w)$ y $\Delta(q_0, w)$ son no vacíos y $\Delta'(q_0, w) \cap \Delta(q_0, w) \neq \emptyset$ (hipótesis de inducción), tenemos que $\Delta(\Delta'(q_0, w) \cap \Delta(q_0, w), a) \subseteq \Delta(\Delta(q_0, w), a) = \Delta(q_0, wa)$, por lo que $\Delta'(q_0, wa) \cap \Delta(q_0, wa) \neq \emptyset$

Lo anterior está garantizando que existe por lo menos un camino para que el procesamiento de la cadena w sea exitoso en el nuevo AFN ■

Ejemplo 6.9 Construir un AFN equivalente al siguiente AFN– λ (Ilustración 6.21 y Tabla 6.10):

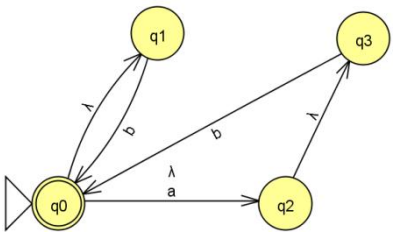

Ilustración 6.21 AFN – λ Ejemplo 6.9

Tabla 6.10 Función Δ Ejemplo 6.9

Δ	a	b	λ
q_0	$\{q_2\}$	$\emptyset$	$\{q_1, q_2\}$
q_1	$\emptyset$	$\{q_0\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_0\}$	$\emptyset$

La Tabla 6.11 muestra las λ- clausuras del AFN– λ:

Tabla 6.11 λ - clausuras Ejemplo 6.9

$\lambda[\{q_0\}] = \{q_0, q_1, q_2, q_3\}$
$\lambda[\{q_1\}] = \{q_1\}$
$\lambda[\{q_2\}] = \{q_2, q_3\}$
$\lambda[\{q_3\}] = \{q_3\}$

Utilizando la Definición 6.10 la función de transición Δ': $Q \times \Sigma \rightarrow \wp(Q)$ es la siguiente:

$\Delta'(q_0, a) = \lambda[\Delta(\lambda[\{q_0\}], a)] = \lambda[\Delta(\{q_0, q_1, q_2, q_3\}, a)] = \lambda[\{q_2\}] = \{q_2, q_3\}$

$\Delta'(q_0, b) = \lambda[\Delta(\lambda[\{q_0\}], b)] = \lambda[\Delta(\{q_0, q_1, q_2, q_3\}, b)] = \lambda[\{q_0\}] = \{q_0, q_1, q_2, q_3\}$

$\Delta'(q_1, a) = \lambda[\Delta(\lambda[\{q_1\}], a)] = \lambda[\Delta(\{q_1\}, a)] = \lambda[\emptyset] = \emptyset$

$\Delta'(q_1, b) = \lambda[\Delta(\lambda[\{q_1\}], b)] = \lambda[\Delta(\{q_1\}, b)] = \lambda[\{q_0\}] = \{q_0, q_1, q_2, q_3\}$

$\Delta'(q_2, a) = \lambda[\Delta(\lambda[\{q_2\}], a)] = \lambda[\Delta(\{q_2, q_3\}, a)] = \lambda[\{\emptyset\}] = \emptyset$

$\Delta'(q_2, b) = \lambda[\Delta(\lambda[\{q_2\}], b)] = \lambda[\Delta(\{q_2, q_3\}, b)] = \lambda[\{q_0\}] = \{q_0, q_1, q_2, q_3\}$

$\Delta'(q_3, a) = \lambda[\Delta(\lambda[\{q_3\}], a)] = \lambda[\Delta(\{q_3\}, a)] = \lambda[\emptyset] = \emptyset$

$\Delta'(q_3, b) = \lambda[\Delta(\lambda[\{q_3\}], b)] = \lambda[\Delta(\{q_3\}, b)] = \lambda[\{q_0\}] = \{q_0, q_1, q_2, q_3\}$

Finalmente se obtiene el autómata pedido (Ilustración 6.22 y Tabla 6.12):

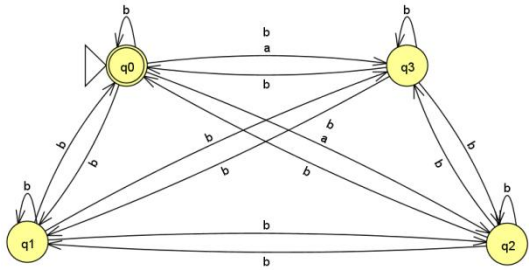

Ilustración 6.22 Nuevo AFN Ejemplo 6.9

Tabla 6.12 Función Δ' Ejemplo 6.9

Δ'	a	b
q_0	$\{q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
q_1	$\emptyset$	$\{q_0, q_1, q_2, q_3\}$
q_2	$\emptyset$	$\{q_0, q_1, q_2, q_3\}$
q_3	$\emptyset$	$\{q_0, q_1, q_2, q_3\}$

Corolario 6.1 Los modelos computacional AFD AFN y AFN – λ son equivalentes, es decir AFD ≡ AFN ≡ AFN – λ.

El siguiente teorema ayudará a demostrar un resultado importante que relaciona los lenguajes regulares y los autómatas AFD AFN y AFN-λ.

Teorema 6.3 (Lema de Arden) Si A y B son lenguajes sobre un alfabeto Σ y $\lambda \notin A$, entonces la ecuación $X = AX \cup B$ tiene una única solución dada por $X = A^*B$.

Demostración: primero se verificará si efectivamente $X = A^*B$ es solución de $X = AX \cup B$. Utilizando el Teorema 5.2, 5.4, 5.3 y la definición de clausura de Kleene se obtiene:

$$X = AX \cup B = A(A^*B) \cup B = (AA^*)B \cup B = A^+B \cup B = (A^+ \cup \lambda)B = (A^*)B = A^*B$$

Supóngase que existe otra solución y esta es $X = A^*B \cup C$ donde $C \cap A^*B = \emptyset$:

$$X = AX \cup B = A(A^*B \cup C) \cup B = (AA^*B \cup AC) \cup B = (A^+ \cup \lambda)B \cup AC = A^*B \cup AC.$$

Intersecando con C a los miembros de los extremos de las igualdades del renglón anterior:

$$X \cap C = (A^*B \cup AC) \cap C$$
$$(A^*B \cup C) \cap C = (A^*B \cup AC) \cap C$$
$$(A^*B \cap C) \cup (C \cap C) = (A^*B \cap C) \cup (AC \cap C)$$
$$C = AC \cap C$$

Se concluye que $C \subseteq AC$. Ahora si $C \neq \emptyset$ existe una cadena de longitud mínima $x \in C$ y como $C \subseteq AC$ se tiene también que $x \in AC$. Como $x \in AC$ se puede reescribir a x como la concatenación de dos cadenas y e z donde necesariamente $y \in A$ y $z \in C$. Debido a que $\lambda \notin A$ se tiene que $y \neq \lambda$, ahora se sabe que $|z| < |x|$. Como se supuso que la cadena x era de longitud mínima y $z \in C$ tiene una longitud menor que x se llega a la conclusión por el principio de no contradicción que $C = \emptyset$ ■

Un resultado importante (Teorema 6.4 y 6.5) que relaciona los autómatas AFD, AFN, AFN – λ con los lenguajes regulares (Definición 5.13) sostiene que un lenguaje es regular si y solo si es aceptado por un autómata finito (AFD, AFN, AFN – λ).

Teorema 6.4 (Teorema de Kleene parte I) Dada una expresión regular R sobre Σ existe un AFN – λ tal que $L(M) = R$.

Este teorema establece que el AFN – λ que se construya sólo puede aceptar las cadenas que de deducen de R.

Demostración: Se razonará inductivamente usando la Definición 5.14 de expresión regular. Se construirán autómatas para las expresiones regulares básicas siguientes:

Un AFN – λ que acepta el lenguaje $R = \emptyset$ sobre el alfabeto Σ, muestra en la Ilustración 6.23.

Un AFN – λ que acepta el lenguaje $R = \{\lambda\}$ se muestra en la Ilustración 6.24.

Ilustración 6.23 AFN – λ que acepta el lenguaje *R=∅*

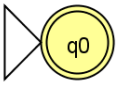

Ilustración 6.24 AFN – λ que acepta el lenguaje *R={λ}*

Un AFN – λ que acepta el lenguaje $R = \{a\}, a \in \Sigma$, se muestra en la Ilustración 6.25.

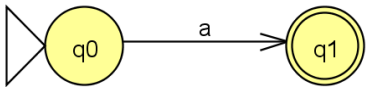

Ilustración 6.25 AFN – λ que acepte el lenguaje *R={a}*

Hipótesis de inducción: Supóngase que para las expresiones regulares R y S existen AFN – λ M_1 y M_2 tal que $L(M_1) = R$ y $L(M_2) = S$. Esquemáticamente se presentarán los autómatas M_1 y M_2 se la siguiente manera (Ilustración 6.26):

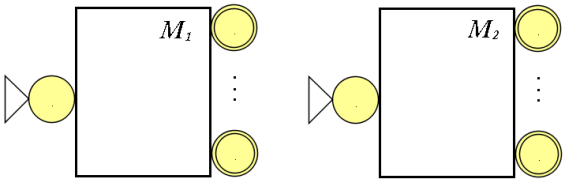

Ilustración 6.26 Autómatas *M1* y *M2*. Fuente (De Castro, 2004, p. 50)

Las ilustraciones presentadas a continuación (Ilustración 6.27, Ilustración 6.28 e Ilustración 6.29) son AFN – λ que aceptan los lenguajes RS , $R \cup S$ y R^* respectivamente:

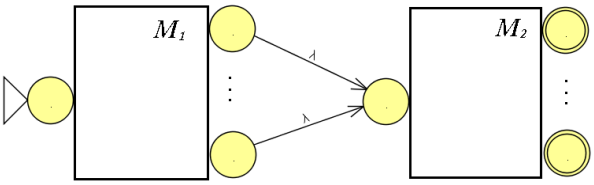

Ilustración 6.27 AFN – λ que acepta el lenguaje *RS*. Fuente (De Castro, 2004, p. 51)

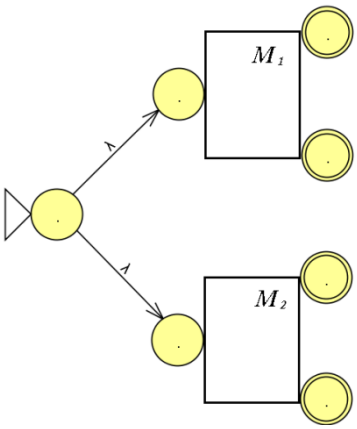

Ilustración 6.28 AFN – λ que acepta el lenguaje *R∪S*. Fuente (De Castro, 2004, p. 51)

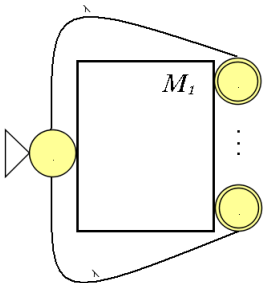

Ilustración 6.29 AFN – λ que acepta el lenguaje *R. Fuente** (De Castro, 2004, p. 52)

■

Ejemplo 6.10 Dado el lenguaje $a^*(b \cup ab^* \cup ab^*a)c^* \cup (a \cup b)(a \cup ac)^*$ sobre $\Sigma = \{a, b, c\}$ diseñar un AFN – λ que lo acepte.

Para diseñar el AFN – λ que acepte el lenguaje pedido es necesario conocer que por cada operación concatenación y unión es necesario establecer una función de transición λ. Como primer y segundo paso se realizan las transiciones que usan un solo símbolo de Σ y la estrella de Kleene. En el tercer paso se establecerá la concatenación de dos elementos, luego tres elementos y así sucesivamente. En el cuarto paso se agruparan las uniones pedidas por el lenguaje y si es necesario se repiten los últimos pasos según sea el caso para obtener el autómata solicitado.

Paso I: en la Ilustración 6.30 se representan AFN – λ que aceptan las expresiones regulares a. b y c respectivamente:

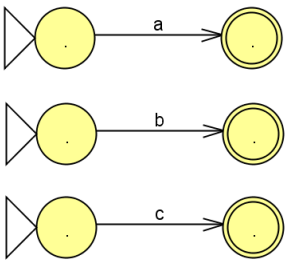

Ilustración 6.30 AFN – λ que aceptan las expresiones *a. b* y *c*

Paso II: en la Ilustración 6.31 se representan AFN – λ que reconocen las expresiones a^*, b^* y c^*. Téngase en cuenta que el autómata que reconoce a^* reconoce también al símbolo a.

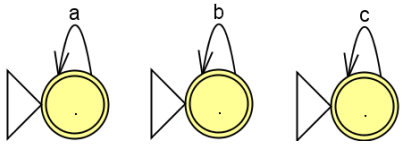

Ilustración 6.31 AFN – λ que aceptan *a*, b y *c****

Paso III: Ahora los siguientes AFN – λ reconocen las expresiones ac, ab^* y ab^*a respectivamente (Ilustración 6.32, Ilustración 6.33 e Ilustración 6.34).

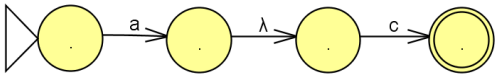

Ilustración 6.32 AFN – λ que reconoce *ac*

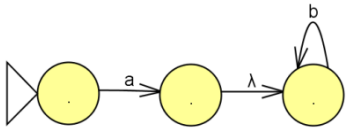

Ilustración 6.33 AFN – λ que reconoce *ab**

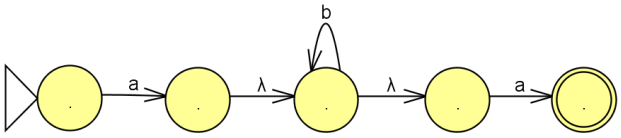

Ilustración 6.34 AFN – λ que reconoce *ab*a*

Paso IV: se representa el AFN – λ que reconoce $a^*(b \cup ab^* \cup ab^*a)c^*$(Ilustración 6.35):

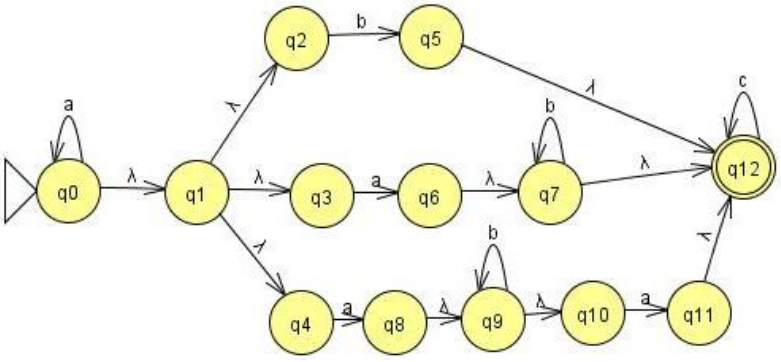

Ilustración 6.35 AFN – λ que reconoce *a* (b∪ab*∪ab* a) c**

El AFN – λ que acepta $(a \cup b)(a \cup ac)^*$ (Ilustración 6.36):

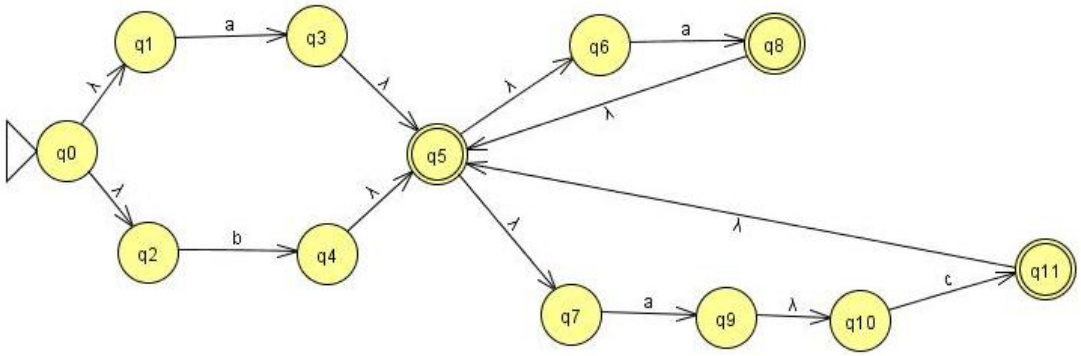

Ilustración 6.36 AFN – λ que acepta *(a∪b) (a∪ac)**

Finalmente el AFN – λ que acepta $a^*(b \cup ab^* \cup ab^*a)c^* \cup (a \cup b)(a \cup ac)^*$ (Ilustración 6.37):

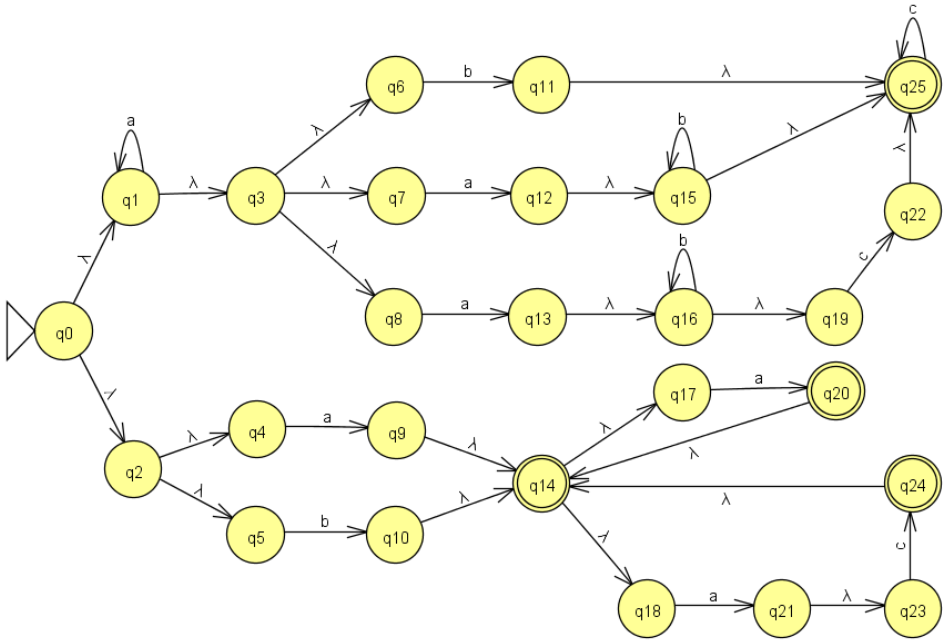

Ilustración 6.37 AFN – λ que acepta *a* (b∪ab*∪ab* a) c*∪(a∪b) (a∪ac)**

Teorema 6.5 (Teorema de Kleene parte II) Dado un AFN se puede encontrar una expresión regular R tal que $L(M) = R$.

Este teorema establece que se puede encontrar una expresión regular de modo tal que las cadenas aceptadas por el AFN sólo sean las que se puedan deducir de R y ninguna otra más.

Demostración: dado un AFN $M = \{\Sigma, Q, q_0, F, \Delta\}$ con estado inicial q_0, se puede a partir de este autómata construir uno nuevo diferente $M_i = \{\Sigma, Q, q_i, F, \Delta\}$ con $q_i \in Q$ y que coincide con M en todos sus parámetros excepto en el estado inicial (M_i acepta cadenas distintas por ende acepta un lenguaje distinto), por lo tanto $L(M_i) = L_i$ y $L(M) = L_0$. Finalmente $L_i = \{w \in \Sigma^* : \Delta(q_i, w) \cap F \neq \emptyset\}$.

En otras palabras $L_i = \{w \in \Sigma^* : \Delta(q_i, w) \cap F \neq \emptyset\}$ nos dice que el lenguaje L_i es igual a todas las cadenas w que finalizan en un algún estado de aceptación. Ahora se puede reescribir a L_i también como:

$$L_i = \begin{cases} \bigcup_{a \in \Sigma} \{aL_j : q_j \in \Delta(q_i, a)\}, si\ q_i \notin F \\ \bigcup_{a \in \Sigma} \{aL_j : q_j \in \Delta(q_i, a)\} \cup\ \lambda, si\ q_i \in F \end{cases}$$

En esta nueva escritura de L_i se reconoce que por cada estado $q_i \in Q$ existirá una ecuación. Por tanto existirá un sistema de $n + 1$ ecuaciones con $n + 1$ incognitas.

$$\begin{cases} L_0 = a_{01}L_0 \cup a_{02}L_1 \cdots \cup a_{0n}L_n \\ L_1 = a_{11}L_0 \cup a_{12}L_1 \cdots \cup a_{1n}L_n \\ \vdots \\ L_n = a_{n1}L_0 \cup a_{n2}L_1 \cdots \cup a_{nn}L_n \end{cases}$$

Donde $a_{ij} \in \Sigma$ o es igual a $\emptyset$ y si la ecuación está asociada a un estado de aceptación se le agrega el símbolo λ.

Se mostrará a continuación que el sistema presentado tiene una única solución. La última ecuación puede ser escrita de la siguiente manera:

$$L_n = a_{nn}L_n \cup a_{n1}L_0 \cup a_{n2}L_1 \cdots \cup a_{n(n-1)}L_{n-1}$$

Aplicando el lema de Arden y luego concatenando respecto a la unión se obtiene:

$$L_n = (a_{nn})^*(a_{n1}L_0 \cup a_{n2}L_1 \cdots \cup a_{n(n-1)}L_{n-1})$$

$$L_n = (a_{nn})^*a_{n1}L_0 \cup (a_{nn})^*a_{n2}L_1 \cdots \cup (a_{nn})^*a_{n(n-1)}L_{n-1}$$

La anterior ecuación está en términos de L_0 hasta L_{n-1} y se remplaza en la ecuación $L_{n-1} = a_{(n-1)1}L_0 \cup a_{(n-1)2}L_1 \cdots \cup a_{(n-1)n}L_n$, la cual también queda en términos de L_0 hasta L_{n-1}, así:

$$L_{n-1} = a_{(n-1)1}L_0 \cup a_{(n-1)2}L_1 \cdots \cup a_{(n-1)n}((a_{nn})^* a_{n1}L_0 \cup (a_{nn})^* a_{n2}L_1 \cdots \cup (a_{nn})^* a_{n(n-1)}L_{n-1})$$

Repitiendo este procedimiento sucesivas veces el sistema se reduce a una sola ecuación con una sola incógnita, en este caso L_0 en términos de L_0. Aplicando el lema de Arden nuevamente se llega a la solución final donde $L(M) = L_0 = R$ ■

Ejemplo 6.11 Utilizando el lema de Arden y el teorema de Kleene, encontrar las expresiones regulares para los lenguajes aceptados por el siguiente AFN (Ilustración 6.38).

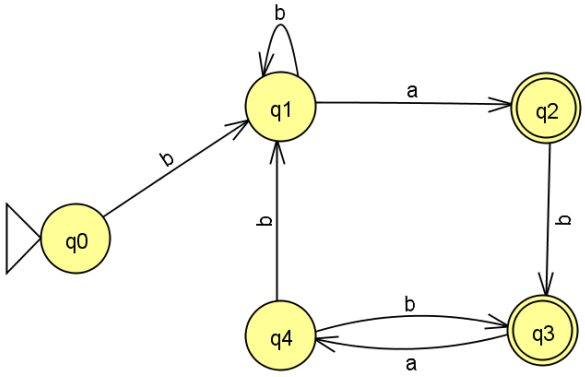

Ilustración 6.38 AFN Ejemplo 6.11

Lo primero es establecer el alfabeto, en este caso $\Sigma = \{a, b\}$. Como segundo se plantean las ecuaciones que están asociadas a cada uno de los estados del AFN. Puesto hay 5 estados entonces será un sistema de 5 ecuaciones con cinco incógnitas y que estarán asociadas a las funciones de transición. Por ejemplo, en q_0 solo existe una transición que procesa el símbolo b, luego $L_0 = bL_1$. Como en el estado q_1 hay dos transiciones su ecuación asociada es $L_1 = aL_2 \cup bL_1$. Obsérvese que el estado q_2 es un estado final y sólo tiene una transición entonces la ecuación asociada es $L_2 = bL_3 \cup \lambda$. Se sigue sucesivamente hasta completar las ecuaciones asociadas a los estados del autómata. Se tiene el siguiente sistema de ecuaciones:

$$\begin{cases} L_0 = bL_1 & (1) \\ L_1 = aL_2 \cup bL_1 & (2) \\ L_2 = bL_3 \cup \lambda & (3) \\ L_3 = aL_4 \cup \lambda & (4) \\ L_4 = bL_3 \cup bL_1 & (5) \end{cases}$$

Se sustituye (5) en (4) y obtenemos $L_3 = a(bL_3 \cup bL_1) \cup \lambda = abL_3 \cup abL_1 \cup \lambda$ y aplicando el lema de Arden (Teorema 6.3):

$$L_3 = (ab)^*(abL_1 \cup \lambda) \quad (6)$$

Sustituyendo (6) en (3) $L_2 = b\big((ab)^*(abL_1 \cup \lambda)\big) \cup \lambda$. Por la ley asociativa de la concatenación de lenguajes $L_2 = b(ab)^*(abL_1 \cup \lambda) \cup \lambda$ y ahora concatenando respecto a la unión se tiene que: $L_2 = b(ab)^*abL_1 \cup b(ab)^* \cup \lambda$. Aplicando el Teorema 5.4:

$$L_2 = b(ab)^+L_1 \cup b(ab)^* \cup \lambda \quad (7)$$

Usando (7) en (2) se obtiene $L_1 = a(b(ab)^+L_1 \cup b(ab)^* \cup \lambda) \cup bL_1$, reorganizando: se tiene que $L_1 = ab(ab)^+L_1 \cup ab(ab)^* \cup a \cup bL_1 = ab(ab)^+L_1 \cup (ab)^+ \cup a \cup bL_1$. Ahora se tiene que $L_1 = (ab(ab)^+ \cup b)L_1 \cup (ab)^+ \cup a$.Por el lema de Arden:

$$L_1 = (ab(ab)^+ \cup b)^*((ab)^+ \cup a) \quad (8)$$

Luego sustituyendo (8) en (1) $L_0 = b((ab(ab)^+ \cup b)^*((ab)^+ \cup a))$. Finalmente:

$$L_0 = b(ab(ab)^+ \cup b)^*((ab)^+ \cup a)$$

donde $L_0 = R$.

Como ejercicio se mostrará a continuación porque la demostración del Teorema 6.5 no es válida para AFN-λ.

Ejemplo 6.12 Encuentre el lenguaje regular aceptado por el AFN-λ de acuerdo al método presentado en la demostración del Teorema 6.5 (Ilustración 6.39):

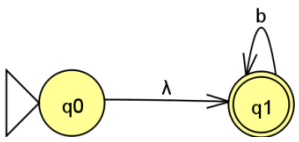

Ilustración 6.39 AFN-λ Ejemplo 6.12

La primera ecuación del sistema es $L_0 = \emptyset$ puesto que en la función de transición estado inicial q_0 no está definido para símbolo alguno. Se incurre en un error si se escribe $L_0 = \lambda L_1 \cup \lambda$ pues λ está siendo tratado como un símbolo y por la Nota 6.3 este no lo es. Continuando se propone el siguiente sistema:

$$\begin{cases} L_0 = \emptyset \\ L_1 = bL_1 \cup \lambda \end{cases}$$

Aplicando el lema de arden a $L_1 = bL_1 \cup \lambda$ se tiene que $L_1 = b^*\lambda = b^*$. Esta última ecuación es imposible aplicar en L_0. Por lo tanto la demostración constructiva del Teorema 6.5 no es validad para AFN-λ, para ello es necesario transformarlo primero en un AFN.

Ejemplo 6.13 En el Ejemplo 6.5 se propuso un AFPD M y que acepta el lenguaje $L = \{0^i 1^{2i} / i \geq 1\}$. Se mostrará porque L no es regular:

Se puede establecer por extensión las cadenas que están en L:

$$L = \{011, 001111, 000111111, 000011111111, \dots\}$$

Se puede construir de manera incremental el conjunto L de este modo:

$$L_1 = \{0^1 1^2\} = \{011\}$$
$$L_2 = \{0^2 1^4\} = \{001111\}$$
$$L_3 = \{0^3 1^6\} = \{000111111\}$$
$$L_n = \{0^n 1^{2n}\} = \{0 \dots 011 \dots 11\}$$
$$\dots$$

Por tanto $L = L_1 \cup L_2 \cup L_3 \cup \dots \cup L_n \cup \dots$ Por la construcción de la demostración teorema de Keene parte I se puede establecer fácilmente un AFN-λ por cada L_i que hace parte de L. Sin embargo la unión de L_i es infinita lo que nos lleva a construir una infinidad de máquinas que al unirlas tendrán una cantidad ilimitada de estados. En conclusión no existe un AFN-λ ni ningún otro autómata finito (sin pila) que reciba exactamente el lenguaje L.

Puede suceder que no se esté convencido del resultado anterior. Supóngase por un momento que el siguiente autómata finito acepta el lenguaje L (Ilustración 6.40).

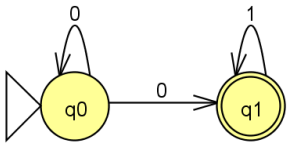

Ilustración 6.40 Autómata finito Ejemplo 6.13

Empero este autómata recibe también cadenas que no están en L, por ejemplo la cadena que tenga un solo cero y un solo uno. Con el siguiente AFN-λ también ocurre lo mismo (Ilustración 6.41):

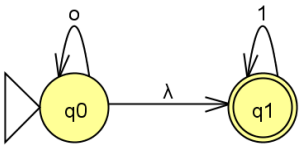

Ilustración 6.41 AFN-λ Ejemplo 6.13

Una manera de detectar si un lenguaje es no regular lo provee el Lema de Bombeo, sin embargo no está dentro de los propósitos de este trabajo mostrarlo.

6.3 Tipos de lenguajes

Hasta el momento sólo se ha profundizado en el estudio de los lenguajes regulares y es gracias al Teorema de Kleene parte i y ii y al Corolario 6.1 que se deduce fácilmente la conexión intima de estos lenguajes con los autómatas AFD AFN y AFN – λ. Sin embargo en el Ejemplo 6.13 se coloca en evidencia que los lenguajes de los ejemplos 6.5 e implícitamente el 6.6 no son regulares, pues estos no son aceptados por los modelos computacionales AFD AFN y AFN – λ, sin embargo si son reconocidos por autómatas AFPD y AFPN.

Los lenguajes aceptados por los AFPD y AFPN corresponden a los denominados lenguajes independientes del contexto, más conocidos como lenguajes LIC. Se puede afirmar que los lenguajes regulares son LIC simplemente construyendo un AFPD que acepte un lenguaje regular y esto se consigue ignorando la pila. Sin embargo la prueba de que no todos los lenguajes LIC no son regulares es brindada en el Ejemplo 6.13.

Noam Chomsky (1928-) lingüista y filósofo estadounidense estableció en 1956 una jerarquía entre diferentes lenguajes que son aceptados por diferentes tipos de máquinas (De Castro, 2004, p. 81). Chomsky clasificó los lenguajes en cuatro tipos los cuales están a su vez asociados a las máquinas que los aceptan (Tabla 6.13).

Tabla 6.13 Tipos de lenguajes

TIPO	LENGUAJES	MÁQUINAS
0	Recursivamente Enumerables	Máquinas de Turing
	Recursivos	
1	Sensibles al Contexto	Autómatas Linealmente Acotados
2	Independientes del Contexto	AFPD y AFPN
3	Regulares	AFD, AFN y AFN-λ

La mayoría de bibliografía consultada omite el estudio de los lenguajes de tipo 1 pues los Autómatas Linealmente Acotados son Máquinas de Turing no deterministas monocinta con leves modificaciones. Otros libros sólo estudian los lenguajes de tipo 3 y luego pasan al tipo 0 y algunos otros inician directamente desde el lenguaje de tipo 0. En la siguiente sección se estudiarán las Máquinas de Turing, los lenguajes que estas aceptan y las relaciones que se pueden establecer entre estos últimos.

6.4 Propiedades de los lenguajes regulares

Las siguientes propiedades formalizan resultados sobre las operaciones concatenación y unión de los lenguajes regulares fruto las definiciones 5.13 y 5.14.

Teorema 6.6 Si L, L_1 y L_2 son lenguajes regulares sobre Σ también lo son:

i. $L_1 \cup L_2$
ii. $L_1 L_2$
iii. L^*
iv. L^+
v. $\bar{L} = \Sigma^* - L$
vi. $L_1 \cap L_2$,
vii. $L_1 - L_2$
viii. $L_1 \Delta L_2$

Demostración:

i. Como L_1 y L_2 son lenguajes regulares, por la Definición 5.13 cada uno de estos lenguajes se forman a partir de los lenguajes básicos $\emptyset$, $\{\lambda\}$, $\{a\}$, para

cada $a \in \Sigma$ utilizando las operaciones unión y concatenación de lenguajes. Nuevamente por la Definición 5.13 $L_1 \cup L_2$ es un lenguaje regular pues este se forma a partir de la unión de dos lenguajes regulares ■

ii. Se demuestra de forma similar a i ■

iii. Se demuestra de forma similar a i ■

iv. Se demuestra de forma similar a i ■

v. Por el teorema de Kleene parte I y los teoremas de equivalencia computacional entre los modelos AFD, ADN y AFN – λ, existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$. Un AFD que acepte el complemento de L se debe intercambiar los estados no finales con los finales. ES decir, si $M' = (\Sigma, Q, q_0, Q - F, \delta)$ entonces $L(M') = \overline{L}$ ■

vi. Se sigue de $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ ■

vii. Se sigue de $L_1 - L_2 = L_1 \cap \overline{L_2}$ ■

viii. Se sigue de $L_1 \Delta L_2 = (L_1 \cup L_2) - (L_1 \cap L_2)$ ■

Teorema 6. 7 Todo lenguaje finito es regular.

Demostración: sea w_i con $n \geq i \geq 1$ una cadena cualquiera de un lenguaje L con una cantidad n de cadenas. Si $L_i = \{w_i\}$ entonces $L = \bigcup_{i=1}^{n}\{w_i\}$. Como cada $\{w_i\}$ es regular, L también lo es ■

Teorema 6.8 Σ^* es un lenguaje regular

Demostración: Como el lenguaje vacío es regular por la Definición 5.13, su complemento Σ^* es regular por el Teorema 6.6 parte v.

Corolario 6.2 La colección $\Re \subseteq \wp(\Sigma^*)$ de todos los lenguajes regulares sobre un alfabeto Σ es un álgebra booleana de conjuntos[22].

[22] Una definición de una álgebra booleana de conjuntos se encuentra en Apostol (2001, p. 573)

7. Máquinas de Turing

Herbert Wise director del documental biográfico *Breaking the Code*, una película basada en la vida de Alan Turing alrededor de su papel en la segunda guerra mundial, muestra en una de las escenas del filme una breve explicación dada por el mismo Alan Turing (Derek Jacobi) cuando es interrogado por un artículo de su autoría publicado en 1936 llamado *On Computable Numbers with an application to the Entscheidungsproblem*. Un fragmento de esta escena traducida al español y pronunciada por Derek Jacobi es el siguiente (Wise, 1996):

> […] Pero la cuestión de la decibilidad estaba aún sin resolver. Hilbert, como había dicho, pensaba que debería existir un *método* claramente definido para decidir de manera clara si o no las afirmaciones matemáticas son demostrables; al problema de la decisión él lo llamó el "Entscheidungsproblem".
>
> En mi trabajo On the Computable Numbers demostré que no puede haber ningún método que funcione para todas las preguntas. Resolver problemas matemáticos requiere una fuente infinita de nuevas ideas. Era, por supuesto, una tarea monumental resolver tal cosa. Era necesario examinar la demostrabilidad de todas las afirmaciones matemáticas del pasado, presente y futuro ¿Cómo podría hacerse eso? Eventualmente una palabra me dio la pista.
>
> La gente había estado hablando sobre la posibilidad de un método mecánico, un método que podría aplicarse mecánicamente a la solución de problemas matemáticos sin ninguna intervención humana o el ingenio
>
> ¡*Máquina*! Ésa era la palabra crucial. Concebí la idea de una máquina, una Máquina de Turing que sería capaz de escanear símbolos matemáticos, para leer si lo desea, una afirmación matemática y llegar al veredicto en cuanto a si o no una afirmación es demostrable. Con este concepto fui capaz de demostrar que Hilbert estaba equivocado ¡Mi idea funcionó!

El concepto de algoritmo se había dado por descontado hasta el momento en aquella época, pues a ningún matemático se la había ocurrido darle una definición formal. En respuesta a esta cuestión Turing propone formalizar este concepto mediante su la famosa Máquina de Turing. En la actualidad el concepto de Máquina de Turing es

aceptado como la formalización del concepto de algoritmo. Según Gallardo et al (2003, p. 41), no se puede demostrar que el concepto de Máquina de Turing es equivalente al concepto intuitivo de algoritmo, sin embargo existen razones que permiten realizar dicha afirmación (Tesis de Church-Turing).

7.1 Definición y funcionamiento de la Máquina de Turing

Definición 7.1 Una Máquina de Turing (MT) M es una 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_o, \mathrm{B}, F)$ donde:

i. Q es un conjunto finito de estados.
ii. Σ es el conjunto de símbolos de entrada que no incluye a B y $\Sigma \subseteq \Gamma$.
iii. Γ es un conjunto de símbolos que conforman el alfabeto de cinta.
iv. δ es la función parcial que determina los movimientos de la máquina, definida como $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, S\}$ donde R representa movimiento a la derecha, L movimiento a la izquierda y S la ausencia de movimiento.
v. $q_o \in Q$ es el estado inicial de la MT.
vi. B es el símbolo que representa la celda vacía tal que $\mathrm{B} \in \Gamma$ y $\mathrm{B} \notin \Sigma$.
vii. F es el conjunto de estados finales, donde $\emptyset \neq F \subseteq Q$.

A modo de ejemplo la transición $\delta(q, a) = (p, b, R)$ significa que la UC escaneando el símbolo a, borra a, luego escribe b, pasa del estado q al p y se mueve a la derecha.

En general una MT procesa una cadena de entrada $w \in \Sigma^*$ desde el estado q_0 colocada en una cinta la cual es infinita en ambas direcciones y donde las demás casillas contienen el símbolo B.

Definición 7.2 **La descripción instantánea** (DI) de una MT M es una expresión de la forma $w_1 q\, w_2$, donde $q \in Q$ es el estado actual de computo de M, $w_1 w_2 \in \Gamma^*$ y la UC está debajo del primer símbolo de la cadena w_2 escaneándolo. Las casillas de la cinta donde no están los símbolos de las cadenas $w_1 w_2$ contienen al símbolo B.

Ejemplo 7.1 La Ilustración 7.1 Descripción instantánea Ejemplo 7.1 representa la DI $0q10101$ de una MT M donde la UC está en el estado q escaneando el carácter 1.

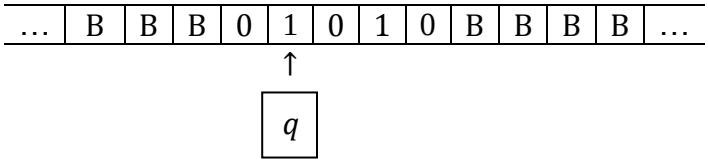

Ilustración 7.1 Descripción instantánea Ejemplo 7.1

Definición 7.3 El **paso computacional** de una MT es el paso de una DI a otra por medio de la función de transición δ la cual se denota por:

$$a_1 \dots a_{i-1} q a_i a_{i+1} \dots a_n \vdash_M a_1 \dots a_{i-1} Y p a_{i+1} \dots a_n$$

con $Y \in \Gamma$ y en la cual se utiliza la transición $\delta(q, a_i) = (p, Y, R)$. Ahora, la notación

$$a_1 \dots a_{i-1} q a_i a_{i+1} \dots a_n \vdash_M^* a_1 \dots a_{j-1} YYYYYYYp a_{j+1} \dots a_n$$

significa que una MT puede pasar de la DI $a_1 \dots a_{i-1} q a_i a_{i+1} \dots a_n$ a la DI $a_1 \dots a_{j-1} YYYYYYYp a_{j+1} \dots a_n$ en n pasos computacionales.

Definición 7.4 Se denomina la **configuración inicial** de una MT cuando la UC se encuentra en el estado q_0 y está situado en el extremo izquierdo de la cadena (escrita en la cinta) bajo el primer símbolo de la misma.

Ejemplo 7.2 La configuración inicial de una MT para la cadena de entrada 000110101 es $q_0 000110101$.

Definición 7.5 El **lenguaje aceptado** por una MT M, es $L(M)$ se define como:

$$L(M) = \{w \in \Sigma^* : q_0 w \vdash_M^* w_1 p w_2, donde\ p \in F\ y\ \ w_1 w_2 \in \Gamma^*\ \}$$

Nota 7.1 Decidir si una cadena es aceptada por una MT demanda menos condiciones a diferencia de los autómatas finitos vistos en el Capítulo 6, pues una cadena no necesariamente debe ser leída en su totalidad para ser aceptada por una MT, sólo se requiere que la máquina se detenga completamente en un momento determinado y en un estado de aceptación. Finalmente cabe agregar que no se permiten transiciones $\delta(q, a)$ cuando $q \in F$.

Ejemplo 7.3 Sea $M = (Q, \Sigma, \Gamma, \delta, q_o, \mathrm{B}, F)$ una MT que acepta el lenguaje no regular L definido por $L = \{0^n 1^n : n \geq 1\}$ cuyas cadenas están conformadas por una cantidad n de ceros seguida también de una cantidad n de unos. Los parámetros de M están definidos a su vez por $Q = \{q_o, q_1, q_2, q_3, q_4\}$, $\Sigma = \{0,1\}$, $\Gamma = \{0,1, X, Y, \mathrm{B}\}$, $F = \{q_4\}$ y la función de transición δ mostrada en la Tabla 7.1.

Tabla 7.1 Función de transición δ Ejemplo 7.3

δ	0	1	X	Y	B
q_o	(q_1, X, R)	$\emptyset$	$\emptyset$	(q_3, Y, R)	$\emptyset$
q_1	$(q_1, 0, R)$	(q_2, Y, L)	$\emptyset$	(q_1, Y, R)	$\emptyset$
q_2	$(q_2, 0, L)$	$\emptyset$	(q_0, X, R)	(q_2, Y, L)	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\emptyset$	(q_3, Y, R)	(q_4, B, R)
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

El diágrafo que representa la MT M se muestra en Ilustración 7.2. Se debe tener en cuenta que el símbolo blanco B en los diágrafos se representará por un cuadro vacío.

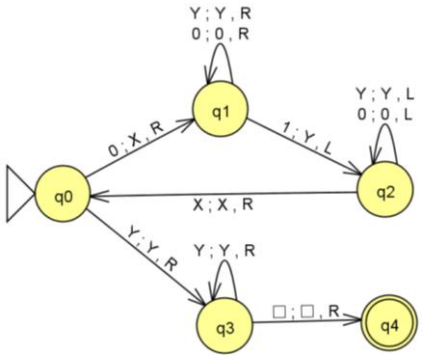

Ilustración 7.2 Diágrafo MT *M* ejemplo 7.3

Una cadena aceptada por M es 000111. La secuencia de descripciones instantáneas desde la configuración inicial y los pasos computacionales es el siguiente:

$$q_o 000111 \vdash Xq_1 00111 \vdash X0q_1 0111 \vdash X00q_1 111 \vdash X0q_2 0Y11 \vdash Xq_2 00Y \vdash q_2 X00Y11 \vdash Xq_0 00Y11 \vdash XXq_1 0Y11 \vdash XX0q_1 Y11 \vdash XX0Yq_1 11 \vdash XX0q_2 YY1 \vdash XXq_2 0YY1 \vdash Xq_2 X0YY1 \vdash XXq_0 0YY1 \vdash XXXq_1 YY1 \vdash XXXYq_1 Y1 \vdash XXXYYq_1 1 \vdash XXXYq_2 YY \vdash XXXq_2 YYY \vdash XXq_2 XYYY \vdash XXXq_0 YYY \vdash XXXYq_3 YY \vdash XXXYYq_3 Y \vdash XXXYYYq_3 \mathrm{B} \vdash XXXYYY\mathrm{B}q_4$$

Una cadena no aceptada por la MT M es 011. La secuencia de descripciones instantáneas es:

$$q_o011 \vdash Xq_111 \vdash q_2XY1 \vdash Xq_0Y1 \vdash XYq_31$$

Nota 7.2 Algunos cómputos que realiza una MT cesan cuando no hay una transición definida. El Ejemplo 7.3 muestra que no existe una transición para la DI XYq_31, pues la MT M se detiene pero no lo hace en un estado de aceptación.

Nota 7.3 Un bucle infinito se representa como $w_1qw_2 \vdash_M^* \infty$ e indica que el cómputo que se inicia con la DI w_1qw_2 no se detiene nunca en la MT. En este caso la MT no puede determinar rechaza o acepta la cadena respectiva.

Nota 7.4 No debe confundirse la transición $\delta(q, \mathrm{B}) = (p, a, R)$ con la transición λ usada para autómatas finitos, pues en este caso y como se ha dicho B es un símbolo y debe ser tratado como tal.

7.2 Lenguajes recursivamente enumerables y recursivos

Definición 7.6 L es un lenguaje **recursivo** (R) si existe una MT M que se para ante cualquier entrada tal que:

i. si $w \in L$ entonces M se para en un estado final.
ii. si $w \notin L$ entonces M se para en un estado no final.

Un lenguaje es R cuando existe una MT que se detiene para todas las cadenas de entrada, bien aceptándolas o bien rechazándolas, no hay otra opción. En otras palabras L es R si existe una MT M tal que $L(M) = L$ y M se detiene con todas las cadenas de entrada.

Ejemplo 7.4 La MT M presentada en el Ejemplo 7.3 es un ejemplo de un lenguaje R.

En la Definición 7.5 se menciona que el lenguaje aceptado por una MT M es aquel que sólo acepta las cadenas para las cuales M se detiene completamente en un momento determinado y en un estado de aceptación, sin embargo, puede suceder que una MT acepte las cadenas de un determinado lenguaje y no sepa qué hacer con algunas otras, cayendo estas en un bucle infinito (Nota 7.4). Por tanto no debe

confundirse el lenguaje aceptado por una MT con un lenguaje R el cual tiene una exigencia adicional (Definición 7.6) y es el hecho de que exista una MT que acepte sus cadenas y rechacé las que no son de él, no hay bucles infinitos.

Definición 7.7 L es un lenguaje **recursivamente numerable** (RE) si existe una MT M tal que:

i. si $w \in L$ entonces M se para en un estado final.
ii. si $w \notin L$ entonces M se para en un estado no final o M no se para.

Corolario 7.1 Si L es un lenguaje recursivo entonces es recursivamente enumerable.

A diferencia de los lenguajes R los lenguajes RE en su definición son más flexibles, pues los RE exigen la existencia de una MT que por lo menos acepte las cadenas de este lenguaje y para las otras es mucho menos exigente, pues no es indispensable que la MT se pare. Más adelante se mostrará que el recíproco del Corolario 7.1 es falso.

Ejemplo 7.5 Un lenguaje recursivamente enumerable es el lenguaje universal denotado por L_u que será mencionado en el capítulo nueve con más detalle.

7.3 Variaciones de Máquinas de Turing

Las MT pueden sufrir modificaciones las cuales no incrementan la capacidad computacional, ni tampoco modifican los lenguajes aceptados por las MT (Gallardo et al., 2003, p. 50). Sin embargo, estas variaciones añaden recursos computacionales que servirán para diseñar algunos algoritmos, los cuales son demasiado complejos de desarrollar en las MT estándar.

7.3.1 Máquinas de Turing con Cinta dividida en Pistas (MT multipista).

Definición 7.8 Una MT multipista es aquella en la que solamente se tiene una UC y una cinta dividida en k pistas, para cualquier valor k finito. La función de transición está definida por $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R, L, S\}$.

La MT multipista lee una tupla de k símbolos, consulta la transición y la realiza, es decir, escribe la nueva tupla de símbolos y cambia de estado moviendo la UC. La función de transición también se puede describir como:

$$\delta\big(q(a_1, a_2, a_3, \dots, a_k)\big) = (p, (b_1, b_2, b_3, \dots, b_k), D)$$

donde a_i y b_i son símbolos de Γ. La Ilustración 7.3 muestra el esquema general de una MT multipista:

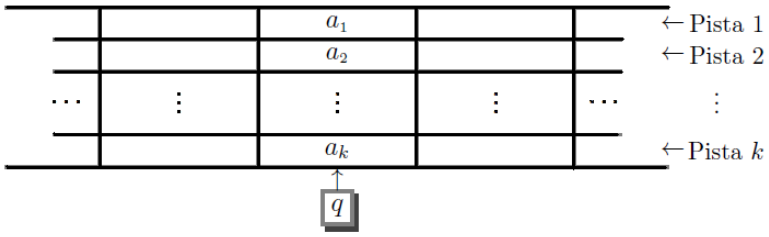

Ilustración 7.3 Esquema general MT multipista. Fuente (De Castro, 2004, p. 181)

Ejemplo 7.6 Las pistas de una MT multipista se usan generalmente para señalar con "marcadores" determinadas posiciones en la cinta. La MT del Ejemplo 7.3 utiliza la idea de marcadores cuando remplaza los ceros por X y los unos por Y. Estos marcadores se pueden colocar en una pista diferente y sin la necesidad de borrar los ceros y unos.

Teorema 7.1 (Equivalencia computacional MT estándar ≡ MT multipista) Dada una MT estándar existe una MT multipista equivalente y viceversa, dada una MT multipista existe una MT estándar equivalente[23].

Demostración parte i: a partir de una MT estándar es fácil construir una MT multipista que acepte exactamente el mismo lenguaje, para ello la UC de la MT multipista debe leer k-tuplas $(s_1, s_2, s_3, \dots, s_k)$ con $s_i \in \Gamma$, donde s_1 es un símbolo de Γ y $s_2, s_3, \dots, s_k$ son símbolos en blanco en todos los cómputos.

Demostración parte i: dada una MT multipista el proceso para construir una MT estándar que acepte el mismo lenguaje de una MT multipista es el siguiente: se debe codificar cada una de las una k-tuplas de la MT multicinta a partir de nuevos símbolos

23 Reformulación: un lenguaje L es aceptado por una MT estándar M si y sólo si es aceptado por una MT multipista M'.

que harán parte del nuevo alfabeto de la cinta MT estándar, para ello no son necesarias nuevas transiciones ■

Ejemplo 7.7 Sea M' una MT multipista con $\Gamma = \{0,1,\mathrm{B}\}$, por tanto las tuplas o el alfabeto de M' es $\Gamma^2 = \{(0,0),(0,1),(0,\mathrm{B}),(1,0),(1,1),(1,\mathrm{B}),(\mathrm{B},0),(\mathrm{B},1),(\mathrm{B},\mathrm{B})\}$. Una MT M estándar que acepta el mismo lenguaje que la MT M' debe tener como alfabeto $\Gamma_M = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$, donde $a_1 = (0,0), a_2 = (0,1), a_3 = (0,\mathrm{B}), a_4 = (1,0), a_5 = (1,1), a_6 = (1,\mathrm{B}), a_7 = (\mathrm{B},0), a_8 = (\mathrm{B},1), a_9 = (\mathrm{B},\mathrm{B})$.

7.3.2 Máquinas de Turing multicinta.

En este modelo de MT existe un número finito de cintas $n \geq 2$ donde cada una de ellas está dividida en celdas al igual que la cinta de una MT estándar. La UC tiene un total de n visores, cada uno para cada cinta (ver Ilustración 7.4).

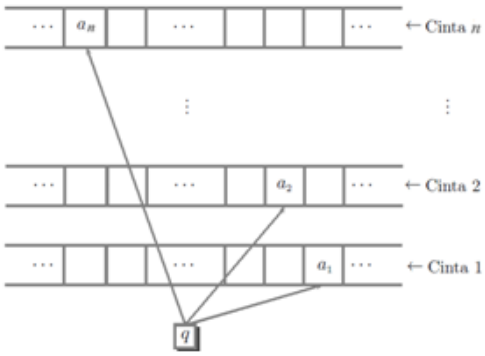

Ilustración 7.4 Esquema general MT multicinta. Fuente (De Castro, 2004, p. 182)

Definición 7.9 **Una MT multicinta** es una MT que consta de una UC, n cintas y n visores para cada una de las cintas, donde $n \geq 2$. La función de transición asociada es:

$$\delta: Q \times \Gamma^{k} \rightarrow Q \times (\Gamma \times \{R, L, S\})^{k}.$$

La función de transición δ también se puede describir como:

$$\delta\big(q, (a_1, a_2, a_3, \ldots, a_n)\big) = \big(p, (b_1, R), (b_2, R), (b_3, R), \ldots, (b_n, R)\big)$$

donde n representa el número de cintas y p es un estado de la MT.

Ejemplo 7.8 A continuación se describirá una MT M multicinta con dos cintas que acepta el lenguaje no regular $\mathrm{L} = \{\mathrm{a}^{\mathrm{i}}\mathrm{b}^{\mathrm{i}}\mathrm{c}^{\mathrm{i}}/\mathrm{i} \geq 0\}$. La máquina M está definida por:

i. $Q = \{q_0, q_1, q_2, q_3\}$.

ii. $q_o \in Q$ es el estado inicial de la MT.

iii. $\Gamma = \{a, b, c, X, \mathrm{B}\}$

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

v. $q_4 \in F$.

vi. Función de transición δ definida en la Tabla 7.2:

Tabla 7.2 Función de transición δ MT multicinta Ejemplo 7.8

$\delta(q_0, (a, \mathrm{B})) = (q_1, (a, R), (X, R))$	$\delta(q_2, (c, \mathrm{B})) = (q_3, (c, S), (\mathrm{B}, R))$
$\delta(q_1, (a, \mathrm{B})) = (q_1, (a, R), (X, R))$	$\delta(q_3, (c, X)) = (q_3, (c, R), (X, R))$
$\delta(q_1, (b, \mathrm{B})) = (q_2, (b, S), (B, L))$	$\delta(q_3, (\mathrm{B}, \mathrm{B})) = (q_4, (\mathrm{B}, S), (\mathrm{B}, S))$
$\delta(q_2, (b, X)) = (q_2, (b, R), (X, L))$	$\delta(q_0, (\mathrm{B}, \mathrm{B})) = (q_4, (\mathrm{B}, S), (\mathrm{B}, S))$

El siguiente es el diágrafo correspondiente a la MT multicinta M (Ilustración 7.5)

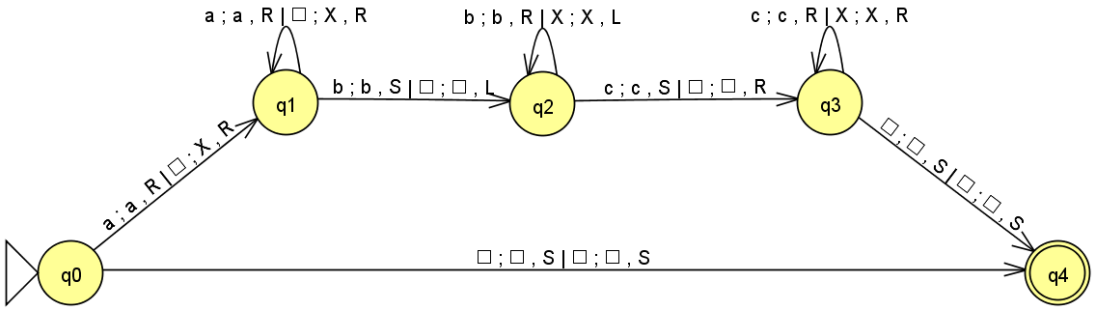

Ilustración 7.5 Diágrafo MT multicinta *M* Ejemplo 7.8

Una cadena aceptada por M es $aabbcc$. La secuencia de descripciones instantáneas en la MT multicinta M es la siguiente: la cadena $aabbcc$ se coloca en la primera cinta y la segunda cinta está llena de blancos. Cada una de las DI mostradas a continuación están separadas por una coma donde la primera corresponde a la primera cinta y la siguiente a la segunda cinta:

$$q_o aabbcc, q_o\mathrm{B} \vdash aq_1 abbcc, Xq_1\mathrm{B} \vdash aaq_1 bbcc, XXq_1\mathrm{B} \vdash aaq_2 bbcc, Xq_2 X \vdash aabq_2 bcc, q_2\mathrm{XX}$$
$$\vdash aabbq_2 cc, q_2\mathrm{BXX} \vdash aabbq_3 cc, q_3 XX \vdash aabbcq_3 c, Xq_3\mathrm{X} \vdash aabbccq_3\mathrm{B}, XXq_3\mathrm{B} \vdash$$

$aabbccq_4\mathrm{B}, XXq_4\mathrm{B}$.

Y puesto que q_4 es un estado de aceptación la cadena es aceptada por M tal como se había dicho. Una cadena no aceptada por la MT multicinta M es aab. La secuencia de descripciones instantáneas es la siguiente:

$$q_o aab, q_o\mathrm{B} \vdash aq_1ab, Xq_1\mathrm{B} \vdash aaq_1b, XXq_1\mathrm{B} \vdash aaq_2b, Xq_2X \vdash aabq_2\mathrm{B}, q_2\mathrm{X}X$$

Como $\delta\big(q_2, (\mathrm{B}, X)\big)$ no está definido la cadena aab no es aceptada por la MT multicinta M.

Teorema 7.2 (Equivalencia computacional MT multipista ≡ MT multicinta) Dada una MT multipista existe una MT multicinta equivalente y viceversa, dado una MT multicinta existe una MT multipista equivalente.

Reformulación: un lenguaje L es aceptado por una MT multipista M si y sólo si es aceptado por una MT multicinta M'.

Demostración parte i: el lenguaje aceptado por una MT multipista M puede también ser aceptado por una MT multicinta M' si los visores de la UC de M' se mueven siempre en el mismo sentido tal como la UC de una MT multipista cualquiera.

Demostración intuitiva parte ii: el lenguaje aceptado por una MT multicinta M' de k cintas es aceptado por una MT multipista del modo siguiente: se construye una MT multipista M con $2k + 1$ pistas, donde por cada cinta de M' hay dos pistas en M, una para el contenido de cada cinta de M' y la otra pista con todas las casillas en blanco, excepto una casilla utilizada como marcador de la posición de la UC en M'. La pista adicional se usa como referencia para los marcadores anteriormente mencionados. Mediante un recorrido de izquierda a derecha la UC sólo captura y almacena la información que se encuentra en los marcadores de las pistas auxiliares hasta tener la información completa para luego mediante un recorrido de derecha a izquierda realizar las modificaciones a las casillas y cambiar los marcadores según sean las transiciones exigidas por la máquina M'.

Por último la función de transición de la máquina M diseñada está definida por:

$$\delta: \left((Q \times \Gamma^{k} \times \{E, M\} \times \{-n, \ldots, 0, 1, \ldots n\}), \Gamma^{2k+1} \right) \rightarrow$$
$$(Q \times \Gamma^{k} \times \{E, M\} \times \{-n, \ldots, 0, 1, \ldots n\}), \Gamma^{2k+1}, \{R, L, S\}.$$

Donde Q es el conjunto de estados de la MT multicinta de la Definición 7.7, Γ el conjunto de símbolos de la máquina, E el símbolo que indica si la UC está en modo lectura y M si la UC se encuentra en modo modificación, R, L, S denotan la dirección del movimiento de la UC ya sea derecha, izquierda y no moverse respectivamente y finalmente el conjunto finito $\{-n, \ldots, 0, 1, \ldots n\}$ cuyos símbolos ayudan a referenciar la marca del visor que se encuentra más a la izquierda en la máquina M ■

La Ilustración 7.6 muestra la simulación de una MT de tres cintas con una MT de siete pistas.

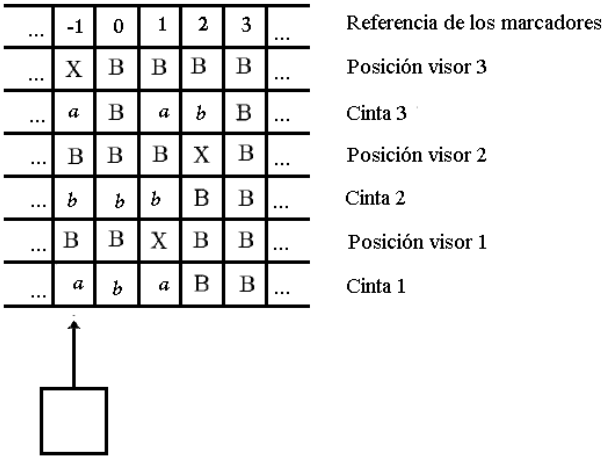

Ilustración 7.6 Simulación MT de tres cintas con una MT de siete pistas

Con el propósito de no dejar vacíos en la demostración de la segunda parte del Teorema 7.3, el Ejemplo 7.9 ilustra y muestra como algunos movimientos de una MT multicinta (Parte i), en lo posible representativos, son simulados por una MT multipista (Parte ii). El elemento central del ejercicio consiste en mostrar que la UC no sólo puede representar el estado y la posición en la cinta sino también almacenar una cantidad finita de datos con el fin de conseguir exitosamente la simulación.

Ejemplo 7.9 (Parte i) La Ilustración 7.7 muestra una MT de dos cintas con las posiciones de los visores de la UC en cada una de las cintas. Recuérdese que los movimientos de los visores son independientes uno al otro.

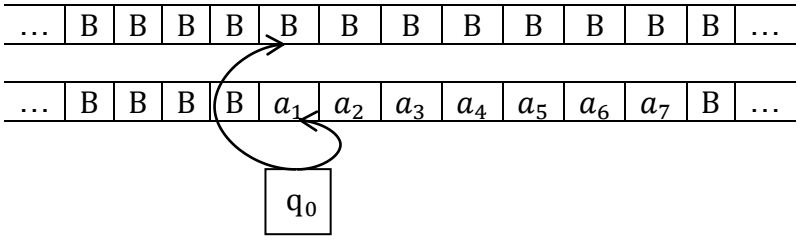

Ilustración 7.7 MT de dos cintas Ejemplo 7.11

En la primera cinta el visor está apuntando a la casilla donde está el primer carácter de la cadena $a_1a_2a_3a_4a_5a_6a_7$, adviértase que los símbolos de esta cadena no necesariamente son todos diferentes y la cantidad de los mismos no es elemento relevante. La segunda cinta está llena de blancos al igual que la MT del Ejemplo 7.11. Por último los visores de la UC se encuentran uno exactamente bajo el otro, sin embargo este tampoco es un factor importante.

Primer movimiento: la UC se encuentra en el estado q_0, el primer visor al leer a_1, lo cambia por b_1, se mueve una casilla a la derecha y queda bajo a_2; el otro visor al leer B, lo cambia por c_1 y se mueve a la izquierda (Ilustración 7.8). La UC cambia al estado q_1.

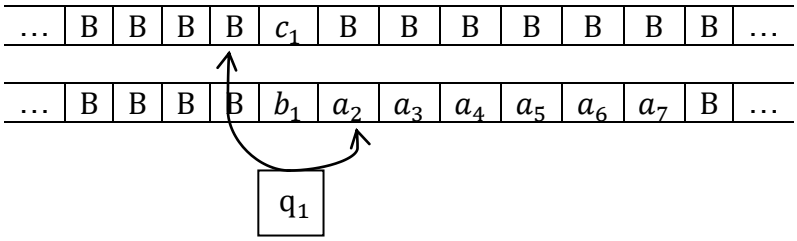

Ilustración 7.8 Primer movimiento MT dos cintas Ejemplo 7.11

El segundo movimiento es el siguiente: el primer visor al leer a_2, lo cambia por b_2, se mueve una casilla a la derecha y queda bajo a_3; el segundo visor al leer B, lo cambia por c_2 y no se mueve (Ilustración 7.9). La UC cambia al estado q_2.

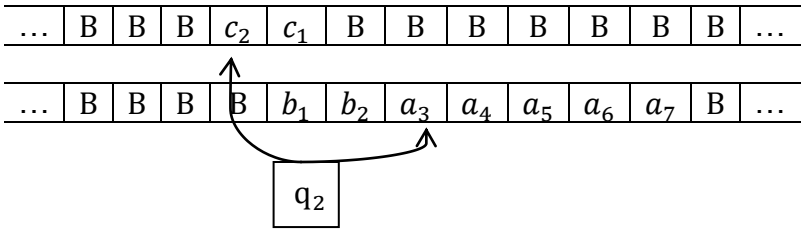

Ilustración 7.9 Segundo movimiento MT dos cintas Ejemplo 7.11

En el último movimiento el primer visor al leer a_3, lo cambia por b_3, se mueve una casilla a la derecha y queda bajo a_4; el segundo visor al leer c_2, lo cambia por c_3 y se mueve a la izquierda (Ilustración 7.10). La UC cambia al estado q_3.

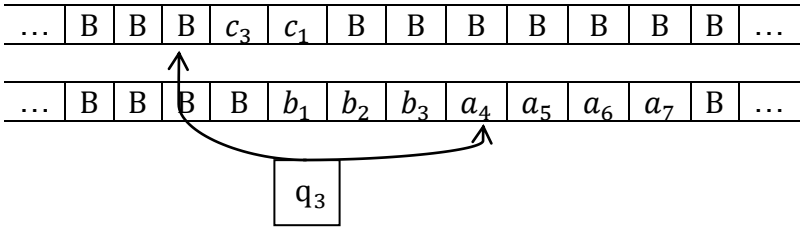

Ilustración 7.10 Tercer movimiento MT dos cintas Ejemplo 7.11

Cabe resaltar que aunque no se definan formalmente las transiciones en una tabla de la MT multicinta, la descripción anterior presentada es prácticamente equivalente.

(Parte ii) De acuerdo a la demostración intuitiva del Teorema 7.2, la MT multipista que simula los movimientos de la MT multicinta se muestra en la Ilustración 7.11.

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	a_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

q_0 , B, B, E, 0

Ilustración 7.11 MT de cinco pistas Ejemplo 7.11

Obsérvese que la UC de la MT multicinta se encuentra en el estado inicial $(\mathrm{q}_0, \mathrm{B}, \mathrm{B}, E, 0) \in (Q \times \Gamma^2 \times \{E, M\} \times \{-n, \ldots, 0, 1, \ldots n\})$, siendo este por definición el único estado de este tipo sin importar que existan otras tuplas que contengan como primer carácter a q_0. El segundo y tercer carácter indica que la UC aún no ha leído los marcadores (X) de la segunda y cuarta pista que indican las posiciones de los visores en la MT multicinta. El cuarto carácter indica que la UC está en modo lectura y el quinto carácter dice que la posición del marcador que se encuentra más a la izquierda (en la segunda y cuarta cintas) está bajo el cero de la quinta pista. A continuación se simulan tres movimientos de la MT multicinta en la MT multipista.

Primer movimiento. Se define la transición:

$$\big((\mathrm{q}_0, \mathrm{B}, \mathrm{B}, E, 0), (a_1, X, \mathrm{B}, X, 0)\big) \mapsto \big((\mathrm{q}_0, X, X, E, 0), (a_1, X, \mathrm{B}, X, 0), S\big)$$

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	a_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, X, X, E, 0$

Ilustración 7.12 Primera transición MT multicinta Ejemplo 7.11

donde la UC cambia de estado, no modifica la tupla leída y no se mueve (Ilustración 7.12). Obsérvese que el nuevo estado, en el segundo y tercer carácter, ya tomó lectura de los marcadores en la MT multicinta. Se pasa a la siguiente transición donde se da fin a la etapa de lectura y el nuevo estado indica el modo modificación (Ilustración 7.13):

$$\big((\mathrm{q}_0, X, X, E, 0), (a_1, X, \mathrm{B}, X, 0)\big) \mapsto \big((\mathrm{q}_0, X, X, M, 0), (a_1, X, \mathrm{B}, X, 0), S\big)$$

	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	a_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, X, X, M, 0$

Ilustración 7.13 Segunda transición MT multicinta Ejemplo 7.11

La tercera transición muestra que la UC modifica la tupla leída por $(b_1, \mathrm{B}, \mathrm{B}, X, 0)$ y se mueve a la derecha. El nuevo estado indica en el segundo carácter que ya realizó la modificación del símbolo de la primera pista y el tercer carácter que aún queda una modificación pendiente por realizar (Ilustración 7.14):

$$\big((\mathrm{q}_0, X, X, M, 0), (a_1, X, B, X, 0)\big) \mapsto \big((\mathrm{q}_0, \mathrm{B}, X, M, 0), (b_1, \mathrm{B}, \mathrm{B}, X, 0), R\big)$$

La transición que sigue indica que la UC coloca el marcador que simula al primer visor de la MT multicinta (Ilustración 7.15):

$$\big((\mathrm{q}_0, \mathrm{B}, X, M, 0), (a_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_0, \mathrm{B}, X, M, 0), (a_2, X, \mathrm{B}, \mathrm{B}, 1), L\big)$$

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	b_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, \mathrm{B}, X, M, 0$

Ilustración 7.14 Tercera transición MT multicinta Ejemplo 7.11

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	X	B	B	B	B	B	B	B	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	B	X	B	B	B	B	B	B	…
…	B	B	B	B	b_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, \mathrm{B}, X, M, 0$

Ilustración 7.15 Cuarta transición MT multicinta Ejemplo 7.11

Por ello la quinta transición es (Ilustración 7.16):

$$\big((\mathrm{q}_0, \mathrm{B}, X, M, 0), (b_1, \mathrm{B}, \mathrm{B}, X, 0)\big) \mapsto \big((\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, 0), (b_1, \mathrm{B}, c_1, B, 0), L\big)$$

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	B	B	c_1	B	B	B	B	B	B	B	…
…	B	B	B	B	B	X	B	B	B	B	B	B	…
…	B	B	B	B	b_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, 0$

Ilustración 7.16 Quinta transición MT multicinta Ejemplo 7.11

La sexta transición (Ilustración 7.17):

$$\big((\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, 0), (\mathrm{B}, \mathrm{B}, \mathrm{B}, \mathrm{B}, -1)\big) \mapsto \big((\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1), S\big)$$

La séptima transición (Ilustración 7.18):

$$\big((\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, \mathrm{B}, E, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1), S\big)$$

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	X	B	B	B	B	B	B	B	B	…
…	B	B	B	B	c_1	B	B	B	B	B	B	B	…
…	B	B	B	B	B	X	B	B	B	B	B	B	…
…	B	B	B	B	b_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_0, \mathrm{B}, \mathrm{B}, M, -1$

Ilustración 7.17 Sexta transición MT multicinta Ejemplo 7.11

…	−4	−3	−2	−1	0	1	2	3	4	5	6	7	…
…	B	B	B	X	B	B	B	B	B	B	B	B	…
…	B	B	B	B	c_1	B	B	B	B	B	B	B	…
…	B	B	B	B	B	X	B	B	B	B	B	B	…
…	B	B	B	B	b_1	a_2	a_3	a_4	a_5	a_6	a_7	B	…

$\mathrm{q}_1, \mathrm{B}, \mathrm{B}, E, -1$

Ilustración 7.18 transición MT multicinta Ejemplo 7.11

Las demás transiciones son las siguientes:

$$\big((\mathrm{q}_1, \mathrm{B}, \mathrm{B}, E, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, E, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1), R\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, E, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, E, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0), R\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, E, -1), (a_2, X, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_1, X, X, E, -1), (a_2, X, \mathrm{B}, \mathrm{B}, 1), S\big)$$

$$\big((\mathrm{q}_1, X, X, E, -1), (a_2, X, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_1, X, X, M, -1), (a_2, X, \mathrm{B}, \mathrm{B}, 1), S\big)$$

$$\big((\mathrm{q}_1, X, X, M, -1), (a_2, X, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1), R\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (a_3, \mathrm{B}, \mathrm{B}, \mathrm{B}, 2)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2), L\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1), L\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0), L\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, X, M, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, X, -1)\big) \mapsto \big((\mathrm{q}_1, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1), S\big)$$

$$\big((\mathrm{q}_1, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, E, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1), S\big)$$

$$\big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, E, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1), R\big)$$

$\big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0), R\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1), R\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2)\big) \mapsto \big((\mathrm{q}_2, X, X, E, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2), S\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, E, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2)\big) \mapsto \big((\mathrm{q}_2, X, X, M, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2), S\big)$

$\big((\mathrm{q}_2, X, X, M, -1), (a_3, X, \mathrm{B}, \mathrm{B}, 2)\big) \mapsto \big((\mathrm{q}_2, X, X, M, -1), (b_3, \mathrm{B}, \mathrm{B}, \mathrm{B}, 2), R\big)$

$\big((\mathrm{q}_2, X, X, M, -1), (a_4, \mathrm{B}, \mathrm{B}, \mathrm{B}, 3)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (a_4, X, \mathrm{B}, \mathrm{B}, 3), L\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_3, \mathrm{B}, \mathrm{B}, \mathrm{B}, 2)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_3, \mathrm{B}, \mathrm{B}, \mathrm{B}, 2), L\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_2, \mathrm{B}, \mathrm{B}, \mathrm{B}, 1), L\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_1, \mathrm{B}, c_1, B, 0)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (b_1, \mathrm{B}, c_1, \mathrm{B}, 0), L\big)$

$\big((\mathrm{q}_2, \mathrm{B}, X, M, -1), (\mathrm{B}, \mathrm{B}, c_2, X, -1)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, c_2, \mathrm{B}, -1), L\big)$

$\big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, M, -1), (\mathrm{B}, \mathrm{B}, \mathrm{B}, \mathrm{B}, -2)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, M, -2), (\mathrm{B}, \mathrm{B}, c_3, X, -2), S\big)$

$\big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, M, -2), (\mathrm{B}, \mathrm{B}, c_3, X, -2)\big) \mapsto \big((\mathrm{q}_2, \mathrm{B}, \mathrm{B}, E, -1), (\mathrm{B}, \mathrm{B}, c_3, X, -2), S\big)$

Con lo que se da fin a la simulación de la MT de dos cintas por medio de una MT de cinco pistas.

7.3.3 Máquinas de Turing no determinista (MTN).

Definición 7.10 Dado un alfabeto Σ la ordenación lexicográfica de Σ^* es una lista ordenada de todas sus cadenas. La ordenación lexicográfica induce un orden en el alfabeto y ordena las cadenas, primero por longitud y luego por la ordenación que corresponda, si son letras por el orden del diccionario y si son dígitos por el valor absoluto de cada uno.

Ejemplo 7.10 Si $\Sigma = \{0,1,2\}$ entonces la ordenación lexicográfica de Σ^* es:

$$\lambda, 0,1,2,00,01,02,10,11,12,20,21,22,000,001, \ldots$$

Definición 7.11 Una MT no determinista (MTN) es una MT compuesta por una cinta y una función de transición no determinista, donde para un par de valores dados: estado actual y símbolo leído, existe un conjunto de posibles ternas. La función de transición δ está dada por:

$$\delta: Q \times \Gamma \rightarrow \wp(Q \times \Gamma \times \{R, L, S\})$$

De este modo una MTN acepta una entrada si y sólo si en alguna de las variaciones de su ejecución se alcanza el estado final.

Teorema 7.3 (Equivalencia computacional MT estándar ≡ MTN) Dada una MT estándar existe una MTN equivalente y viceversa, dado una MTN existe una MT estándar equivalente[24].

Demostración parte i: Una MT M estándar es un caso particular de una MTN M' ya que la función de transición δ de la máquina M siempre proporciona un resultado o ninguno.

Demostración parte ii: para o construir una MT M estándar que acepte el mismo lenguaje de una MTN M', se establece a partir de la función de transición δ (o tabla de transiciones) de la MTN M' un número n el cual corresponde al máximo de elementos de algún conjunto $\delta(q_i, a_i) = \{(q_1, a_1, R_1), (q_2, a_2, R_2), (q_3, a_3, R_3), \dots, (q_n, a_n, R_n)\}$ con $a_i \in \Gamma$ y $q_i \in Q$. Las demás transiciones con un número inferior de opciones se completarán con elementos de ella misma, sin importar si son repetidos, hasta que todos los conjuntos $\delta(q_i, a_i)$ tengan el mismo número de elementos n. Por último también se debe establecer un número k de parejas (q_i, a_i) para las cuales la función de transición tenga sentido y a cada una asignarle un número del 1 al k. En la Tabla 7.3 se presenta la función de transición la MTN M'; el número n corresponde al número de columnas de la matriz excepto la primera y el número k corresponde al número de filas de la misma.

Tabla 7.3 Función de transición MTN M' Teorema 7.3

$\delta(q_{10}, a_{10})$	(p_{11}, b_{11}, R_{11})	(p_{12}, b_{12}, R_{12})	(p_{13}, b_{13}, R_{13})					
$\delta(q_{20}, a_{20})$	(p_{21}, b_{21}, R_{21})	(p_{22}, b_{22}, R_{22})	(p_{23}, b_{23}, R_{23})	(p_{24}, b_{24}, R_{24})	…	(p_{2j}, b_{2j}, R_{2j})	…	(p_{2n}, b_{2n}, R_{2n})
…	…	…	…	…	…	…	…	…
$\delta(q_{k0}, a_{k0})$	(p_{k1}, b_{k1}, R_{k1})	(p_{k2}, b_{k2}, R_{k2})	(p_{k3}, b_{k3}, R_{k3})	(p_{k4}, b_{k4}, R_{k4})	…	(p_{kj}, b_{kj}, R_{kj})		

Ahora se construye una MT N de cuatro cintas donde la primera cinta almacena las cadenas de entrada que acepta o rechaza la MTN M'; la segunda y tercera cinta

[24] Un lenguaje L es aceptado por una MT básica M si y sólo si es aceptado por una MTN M'.

contiene todas las cadenas sobre el alfabeto $\{1,2,\ldots,8,9,\alpha,\beta,\gamma\ldots,k\}$ y $\{1,2,\ldots,8,9,\alpha,\beta,\gamma\ldots,n\}$respectivamente y ordenadas lexicográficamente[25] y separadas por un carácter especial una de la otra y la cuarta cinta llena de blancos (ver Ilustración 7.19).

…	B	B	B	B	B	B	B	B	B	B	B	B	…
…	B	B	1	∗	2	∗	1	1	∗	1	2	∗	…
…	B	B	1	∗	2	∗	1	1	∗	1	2	∗	…
…	B	B	a_1	a_2	a_3	a_4	…	a_i	B	B	B	B	…

q_0

Ilustración 7.19 Configuración cintas MT *N* Teorema 7.3

El funcionamiento de la máquina N es el siguiente: cada cadena de la cinta dos y tres representa un camino u opción en la secuencia de ejecución de la máquina M', por ejemplo, supóngase que la cadena 2141 se encuentra en la cinta dos y la cadena 321 en la cinta tres y los visores respectivos las leen. En este caso la MT N según la cadena 2141 de la segunda cinta tomará las pareja (q_{20}, a_{20}) como estado inicial, luego (q_{10}, a_{10}) como el siguiente estado para la ejecución, después (q_{40}, a_{40}) y por último nuevamente (q_{10}, a_{10}); en el caso de que (q_{20}, a_{20}) no sea el estado inicial en esta configuración o que $\delta(q_{20}, a_{20})$ no envié al estado q_{10} (en la tabla de transiciones de la MT N) o $\delta(q_{10}, a_{10})$ no envié al estado q_{40} y así sucesivamente, no se obtendrá un resultado exitoso, en caso contrario o exitoso la cinta tres a partir de la cadena 321 tomará del conjunto $\delta(q_{20}, a_{20})$ la tercera opción, de la pareja $\delta(q_{10}, a_{10})$ la segunda opción y de la pareja $\delta(q_{40}, a_{40})$ la primera. De la cuarta pareja no se toma ninguna opción.

Si una cadena es aceptada por la máquina M entonces existirá un cómputo que se detiene en un estado de aceptación en la máquina N de acuerdo a la construcción presentada. Lo mismo ocurre en caso contrario.

[25] En el Ejemplo 8.13 se muestra una MT estándar que genera cadenas de unos y ceros en orden lexicográfico.

Por el Teorema 7.2 y Teorema 7.1 existe una MT M que acepta en mismo lenguaje de la MT N ■

Ejemplo 7.11 A partir de la siguiente MTN M´ (Ilustración 7.20) se mostrará intuitivamente como una MT N de cuatro cintas acepta el mismo lenguaje.

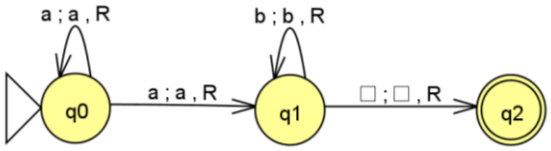

Ilustración 7.20 MTN Ejemplo 7.11

La tabla de transición de la MTN M´ se presenta en la Tabla 7.4:

Tabla 7.4 Función de transición δ Ejemplo 7.11

$\delta(q_0, a)$	(q_0, a, R)	(q_1, a, R)
$\delta(q_1, b)$	(q_1, b, R)	
$\delta(q_1, B)$	(q_2, B, R)	

De acuerdo a la demostración del Teorema 7.3 se establece que $n = 2$ y $k = 3$. La Tabla 7.5 muestra la forma la distribución de las numeraciones y se completan las celdas vacías de la Tabla 7.4:

Tabla 7.5 Nueva presentación de la función de transición δ Ejemplo 7.12

		1	2
1	$\delta(q_0, a)$	(q_0, a, R)	(q_1, a, R)
		1	2
2	$\delta(q_1, b)$	(q_1, b, R)	(q_1, b, R)
		1	2
3	$\delta(q_1, B)$	(q_2, B, R)	(q_2, B, R)

Una cadena aceptada por la MTN es a, las DI son las siguientes:

$$q_0aBB \vdash aq_1BB \vdash aBq_2B$$

Un cómputo no exitoso es:

$$q_0a \vdash aq_0B$$

Obsérvese que la MTN M' acepta la cadena a si el orden de las parejas es (q_0, a), (q_1, B) y si el orden de las opciones es (q_1, a, R), (q_2, B, R) respectivamente, es decir si el visor de la cinta dos lee la cadena 13 y el visor de la cinta tres lee 21.

Corolario 7.2 Los modelos computacional MT estándar, MT multipista y MT multicinta son equivalentes, es decir MT estándar ≡ MT multipista ≡ MT multicinta.

7.3.4 Máquinas de Turing como generadoras de Lenguajes.

Las MT pueden también pueden trabajar como generadoras de cadenas, Para el proceso de generar un lenguaje no son necesarios en la MT los estados de aceptación, sólo el inicial.

Definición 7.12 Una MT $M = \{Q, \Sigma, \Gamma, \delta, q_o, \mathrm{B}\}$ genera un lenguaje $L \subseteq \Sigma^*$ si se cumplen las siguientes dos condiciones:

i. M comienza a operar con la cinta en blanco en el estado inicial q_o.
ii. Cada vez que M retorna al estado inicial q_o hay una cadena del lenguaje L escrita sobre la cinta.

Ejemplo 7.12 La siguiente MT (Ilustración 7.21) genera pares de unos sobre el alfabeto $\Sigma = \{1\}$, en otras palabras el lenguaje representado por $L(M) = \{1^{2i} / i \geq 1\}$

Ilustración 7.21 MT que genera pares de unos Ejemplo 7.6

Ejemplo 7.13 La MT de la Ilustración 7.22 genera cadenas de unos y ceros en orden lexicográfico.

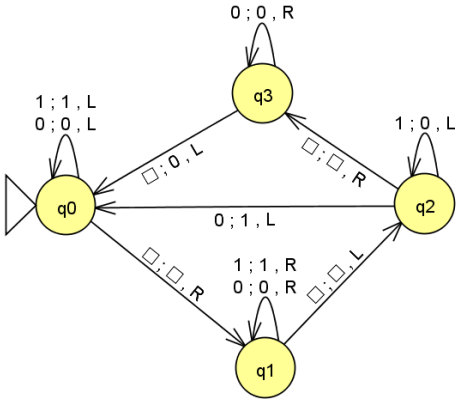

Ilustración 7.22 MT que genera cadenas de ceros y unos en orden lexicográfico Ejemplo 7.14

A continuación se mostrará como las MT generadoras de lenguajes ayudan a caracterizar los lenguajes R y RE.

Teorema 7.4 L es un lenguaje recursivo si y sólo si es generable en orden lexicográfico.

Demostración parte i (sí L es un lenguaje R entonces es generable en orden lexicográfico): sea L un lenguaje R tal que $L = L(M)$ donde M es una MT que acepta el lenguaje L y que se detiene aceptando o rechazando cada una de las cadenas de entrada (la existencia de M está garantizada por la Definición 7.6). Sea M' una MT de dos cintas en la que en la primera se van generando las cadenas de Σ^* separadas por un carácter especial en orden lexicográfico y la otra cinta está llena de blancos. El funcionamiento de M' es el siguiente: cuando el visor de la primera cinta escribe un una cadena de Σ^* inmediatamente M' la copia en la segunda cinta y hace el movimiento necesaria para leerla, en este momento M' simula a M, si la cadena es aceptada por M entonces M' escribe un carácter especial en la cinta dos como separador para escribir la siguiente cadena proporcionada por la cinta uno, si la cadena no es aceptada por M entonces M' la borra y escribe la siguiente cadena proporcionada por la cinta uno. Las cadenas que van quedando en la cinta dos están ordenadas en orden lexicográfico.

Demostración parte ii (Si L es generable en orden lexicográfico entonces L es un lenguaje recursivo): M' genera a L en orden lexicográfico. Se construye una máquina M que acepte a L de la siguiente forma: dada la entrada w, M' debe generar las cadenas de L hasta que esta genera a w o hasta que genere una palabra posterior en orden lexicográfico a w. En el primer caso M acepta y en el segundo caso rechaza ■

Definición 7.13 Se define la ordenación de pares de números de la siguiente forma:

$$(0,0), (0,1), (1,0), (0,2), (1,1), (2,0), (0,3), (1,2), (2,1), (3,0), (0,4), (1,3), (2,2), (3,1), (4,0) \ldots$$

donde los pares (i, j) se ordenan por el valor de la suma $i + j$ y por el valor creciente de i.

El siguiente teorema garantiza la enumerabilidad de los lenguajes RE.

Teorema 7.5 L es un lenguaje recursivamente enumerable si y sólo si existe una MT generadora.

Demostración parte i (si L es un lenguaje RE entonces existe una MT generadora): si L es un lenguaje RE entonces existe una MT M que acepta sus cadenas por la Definición 7.7. Se construirá una máquina M' que genere todas las cadenas de L de cualquier manera o en cualquier orden. M' es una MT de tres cintas, en la primera va generando un por una las cadenas de Σ^* en orden lexicográfico, cada una numerada antes del primer carácter y separadas por un símbolo especial. La cinta dos contiene todas las parejas (i, j) de la definición anterior. Con la tercera cinta M' simula a M.

El lenguaje L se genera del siguiente modo: por cada pareja (i, j) con la que el visor dos se topa, busca en la cinta uno respectiva la cadena i de Σ^*. Luego la máquina M' ejecuta o simula la máquina M en j pasos computacionales, de no ser aceptada la cadena es borrada en la tercera cinta y se continúa con la siguiente pareja. Con la numeración de la cinta dos se está evitando que en M' al simularse una cadena de Σ^* en la cinta tres se entre en un bucle infinito.

Demostración parte ii (Si L es generado por una MT entonces L es un lenguaje recursivamente enumerable): sea M' una MT que genera el lenguaje L, se construirá una MT M que reconozca el lenguaje L tan sólo aceptando las cadenas que le pertenecen. Dada una cadena $w \in \Sigma^*$ generada por M' simplemente M acepta la cadena en caso contrario si $w \in \Sigma^*$ y no es generada por M' simplemente M nunca aceptará dicha cadena, es decir, si $w \notin L$ y L es un conjunto infinito M' nunca generará a w con lo que M nunca terminará su ejecución y no podrá rechazar la cadena. Por tanto L es un lenguaje RE ■

7.3.5 Máquinas de Turing como simuladores de Autómatas Finitos.

Sea $M = (Q, q_0, F, \Sigma, \delta)$ un AFD, ahora a partir de este autómata finito se construirá una MT M' tal que $L(M) = L(M')$. A M' se le añade un nuevo estado q_f que será el único estado de aceptación de M', luego se le agregan transiciones a M' desde los

estados que están en F hasta q_f utilizando el símbolo blanco. Finalmente $M' = (Q', q_0, F', \Sigma, \Gamma, \mathrm{B}, \delta')$ donde:

- $Q' = Q \cup \{q_f\}$
- $\Gamma = \Sigma \cup \{\mathrm{B}\}$
- $F' = \{q_f\}$
- $\delta'(q, s) = (\delta(q, s), s, R)$, para $q \in Q, s \in \Sigma$
- $\delta'(q, \mathrm{B}) = (q_f, \mathrm{B}, S)$, para todo $q \in F$.

Luego la M' se detiene con cualquier entrada w que es aceptada por M. De esta manera se ha demostrado el Teorema 7.6.

Teorema 7.6 Si L es un lenguaje regular entonces es recursivo.

Para mostrar que es falso es recíproco del Teorema 7.6, es decir, si L es un lenguaje recursivo entonces L es un lenguaje regular hay que remitirse al Ejemplo 7.3 y el Ejemplo 6.13 donde se muestra la imposibilidad de que una Lenguaje Independiente del Contexto sea regular.

7.3.6 Máquinas de Turing como simuladores de Autómatas Finitos con Pila.

Sea un AFPD $M = (Q, q_0, F, \Sigma, \Gamma, Z, \Delta)$, a partir de este autómata se construye una MT M' que tenga dos cintas. La primera cinta de M' simula la cinta de entrada de M y en la segunda cinta de M' la pila de M también. Se define $M' = (Q', q_0, F', \Sigma, \Gamma', \mathrm{B}, \delta)$ a través de:

- $Q' = Q \cup \{q_f\}$, donde q_f es un estado nuevo.
- $\Gamma' = \Sigma \cup \Gamma \cup \{\mathrm{B}\}$
- $F' = \{q_f\}$

Ahora se describirá como M' procesa una cadena w desde el inicio del su cómputo. Luego de que la cadena $w = a_1 a_2 \ldots a_k$ se encuentra sobre la primera cinta, lo primero

que debe hacer M' es colocar el símbolo inicial de la pila (Z) sobre la cinta dos (que está llena de blancos) por medio de la transición:

$$\delta\big(q_0,(a_1,\mathrm{B})\big)=\big(q_0,(a_1,S),(Z,\mathrm{S})\big) \text{ para } a_1 \in \Sigma.$$

Así entonces una transición de la forma $\Delta(q,a_1,Z)=(p,\gamma)$ de M se simula con diversas transiciones para lograr que M' sobrescriba la cadena γ en la cinta dos. De este modo para no alterar el lenguaje aceptado por M' cada transición requiere la adición de un estado especial q_e diferente a los demás. Continuando con el ejemplo para la cadena w y la cadena $\gamma = Z_1Z_2 \ldots Z_j$ en la pila, la transición $\Delta(q,a_1,Z)=(p,\gamma)$ se simula del siguiente modo:

$$\delta\big(q_0,(a_1,\mathrm{B})\big)=\big(q_e,(a_1,S),(Z,\mathrm{S})\big)$$

$$\delta\big(q_e,(a_1,\mathrm{Z})\big)=\Big(q_e,(a_1,S),\big(Z_j,R\big)\Big)$$

$$\delta\big(q_e,(a_1,\mathrm{B})\big)=\Big(q_e,(a_1,S),\big(Z_{j-1},R\big)\Big)$$

…

$$\delta\big(q_e,(a_1,\mathrm{B})\big)=\big(p,(a_1,R),(Z_1,S)\big)$$

Ahora se procesa el símbolo siguiente a a_1 que es a_2. La transición de M correspondiente es $\Delta(p,a_2,Z_1)=(p,\mu)$, con $\mu = X_1X_2 \ldots X_m$ cuyos símbolos pertenecen al alfabeto de la pila.

$$\delta\big(p,(a_2,Z_1)\big)=\Big(q_{e_1},(a_2,S),(X_m,R)\Big)$$

$$\delta\Big(q_{e_1},(a_2,\mathrm{B})\Big)=\Big(q_{e_1},(a_2,S),(X_{m-1},R)\Big)$$

…

$$\delta\Big(q_{e_1},(a_2,\mathrm{B})\Big)=\big(p_1,(a_2,R),(X_1,S)\big)$$

Y así sucesivamente hasta completar la lectura de la cadena w. En caso de que se necesite borrar los símbolos de la pila, se debe aplicar la transición $\Delta(p,a_i,A)=(p,\lambda)$, hasta borrar el tope de la pila agregando símbolos en blanco en la segunda cinta de M', por medio de varios desplazamientos hacia la izquierda.

Se ha demostrado que el comportamiento de un AFPD se puede simular con un MT multicinta. Del mismo modo, aunque con un poco más de escritura, se demuestra que

el comportamiento de un AFPN se puede simular con una MT. Se ha demostrado el siguiente teorema:

Teorema 7.7 Todo lenguaje independiente del contexto (LIC) es recursivo.

7.3.7 La Máquina de Turing como computador de funciones.

El siguiente ejemplo muestra como una MT tiene la capacidad de transformar cadenas de entrada:

Ejemplo 7.14 Sea $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$, $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\mathrm{B}\}$, $F = \{q_6\}$ y la tabla de transición presentada en la Tabla 7.6.

Tabla 7.6 Tabla de transición δ Ejemplo 7.14

δ	0	1	B
q_0	(q_5, B, R)	(q_1, B, R)	
q_1	$(q_2, 0, R)$	$(q_1, 1, R)$	
q_2	$(q_2, 0, R)$	$(q_3, 0, I)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, I)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	$(q_4, 1, I)$	$(q_6, 1, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)

La operación $3 - 1$ se ejecuta mediante las siguientes DI:

$$\mathrm{B}q_o 11101\mathrm{B} \vdash \mathrm{B}q_1 1101\mathrm{B} \vdash \mathrm{B}1q_1 101\mathrm{B} \vdash \mathrm{B}11q_1 01\mathrm{B} \vdash \mathrm{B}11q_1 00\mathrm{B} \vdash \mathrm{B}1q_1 100\mathrm{B} \vdash$$
$$\mathrm{B}q_1 1100\mathrm{B} \vdash q_1\mathrm{B}1100\mathrm{B} \vdash \mathrm{B}q_1 1100\mathrm{B} \vdash \mathrm{BB}q_1 100\mathrm{B} \vdash \mathrm{BB}1q_1 00\mathrm{B} \vdash \mathrm{BB}10q_1 0\mathrm{B} \vdash$$
$$\mathrm{BB}10q_1 0\mathrm{B} \vdash \mathrm{BB}100q_2\mathrm{B} \vdash \mathrm{BB}10q_2 0\mathrm{B} \vdash \mathrm{BB}1q_2 0\mathrm{B} \vdash \mathrm{BB}q_2 1\mathrm{B} \vdash \mathrm{B}q_2\mathrm{B}1 \vdash q_6\mathrm{B}11$$

La cadena 11 de la última DI corresponde al resultado de $3 - 1 = 2$.

Una operación binaria como la presentada en el Ejemplo 7.14 es posible notarla funcionalmente como $f(3,1) = 2$ o también en notación binaria $f(11101) = 11$. Así pues, las Máquinas de Turing se pueden utilizar como mecanismos para calcular funciones, es decir una MT M calcula una función $f: \Sigma^* \longrightarrow \Gamma^*$ ya sea total o parcial, si para toda entrada w se tiene que $q_0 w \vdash_M^* q_f v$ con q_0 y q_f estado inicial y final respectivamente, $w \in \Sigma^*$ y $v \in \Gamma^*$, donde $f(w) = v$ y la forma más usada para representar un número n es a través de una sucesión de unos es decir $n \to 1^n$.

Formalmente una función f puede tener j argumentos: $n_1, n_2, \dots, n_j$, estos serían los datos de entrada, cada uno separado del otro por el símbolo "0" así: $1^{n_1}01^{n_2}01^{n_3} \dots 01^{n_j}$. El resultado de la función sería la cadena 1^m escrita sobre la cinta y la función se debe interpretar como $f(n_1, n_2, \dots, n_j) = m$ la cual que será computada en binario por la MT[26].

Definición 7.14 (Función Parcial) Una función f, $f: A \to B$ se dice que es una Función Parcial si existe un subconjunto no vacío de A en el que todos los elementos tienen imagen calculable.

Definición 7.15 (Función Total) Una función f, $f: A \to B$ se dice que es una Función Total si para todo elemento del dominio se puede calcular la imagen.

Definición 7.16 Una función f computada por una MT se denomina función *Turing-computable*, esta función puede ser parcial o total, si una MT computa una función f de j argumentos.

Definición 7.17 Si f computada por la MT está definida para todas las combinaciones de valores de argumentos *j,* se dice que f es una función *Turing-computable total*.

Un resultado importante de la teoría de la computación es el siguiente (García & Martínez, 2005, pp. 123-124):

> La clase de los lenguajes recursivos se puede identificar con la clase de las funciones recursivas totales, mientras la clase de los lenguajes recursivamente enumerables con la clase de las funciones recursivas parciales […] una función total puede ser representada mediante un lenguaje recursivo y una función parcial mediante un lenguaje recursivamente enumerable.

No se pretende ahondar más acerca de esta cuestión, sin embargo por el lado de las Funciones Totales y Parciales Alonzo Church llega a misma conclusión que Alan

[26] Los argumentos de la función y la imagen de la misma en principio pueden ser números naturales, sin embargo hay razón para que los números enteros no puedan serlo, todo depende del alfabeto de la Máquina de Turing que se programe y de la cantidad de símbolos que se quiera usar.

Turing, que el Entscheidungsproblem no tiene solución ya que hablar de Función Computable (funciones parciales y totales) y función Turing-computable es lo mismo[27].

7.4 Más Teoremas sobre lenguajes R y RE

Teorema 7.8 El complemento de un lenguaje recursivo también es recursivo.

Demostración: Sea L un lenguaje R aceptado por una MT M. Se sabe por la Definición 7.6 que M existe y se detiene por completo con cualquier cadena, ya sea para aceptarla o para rechazarla. A partir de las cadenas rechazadas, se puede construir una MT M' que acepte el lenguaje $\bar{L}$ permitiendo que los estados de aceptación de M sean aquellos para los cuales las cadenas son rechazadas en M' y los de rechazo para aquellas cadenas las cuales son aceptadas ■

Teorema 7.9 La unión de lenguajes recursivos es recursivo.

Demostración: Sean L_1 y L_2 dos lenguajes recursivos aceptados por dos MT M_1 y M_2 respectivamente. Se construirá una MT M con dos cintas que simule a M_1 en la primera y a M_2 en la segunda. Dada un entrada cualquiera a M, si M_1 acepta, M aceptará también, de lo contrario la entrada se procesa en la segunda cinta y si M_2 acepta, M acepta, de lo contrario la cadena es rechazada por M ■

Teorema 7.10 La unión de lenguajes RE es RE.

Demostración: El procedimiento de la demostración del Teorema 7.2 no es posible aplicarlo porque puede suceder que M_1 y M_2 nunca se detengan al procesar un entrada en particular, para ello la nueva máquina M debe simular simultáneamente a M_1 y a M_2, donde la existencia de M_1 y M_2 está garantizada por la Definición 7.7.

Se construye a M con dos cintas donde la primera simula a M_1 y la segunda a M_2, sin embargo cada paso computacional de M simulará un paso de M_1 y otro de M_2. Los

[27] El modelo propuesto por Alonzo Church con la ayuda de Kleene es llamado λ-calculus y es un lenguaje formal que permite expresar funciones matemáticas, en este caso funciones computables. El modelo de Church es equivalente al de las máquinas de Turing, esto se demostró en 1936 por el mismo Alan Turing.

estados de M son de la forma (p_i, q_j), donde p es un estado de M_1 y q es un estado de M_2, por lo que las transiciones del tipo:

$$\delta(p_1, s_1) = (p_2, s_2, R_2) \text{ de la } M_1 \text{ y}$$
$$\delta(q_1, r_1) = (q_2, r_2, R_3) \text{ de la } M_2.$$

Corresponde a la transición en M:

$$\delta\big((p_1, q_1), (s_1, r_1)\big) = ((p_2, q_2), (s_2, R_2), (r_2, R_3))$$

Si p_1 o q_1 es un estado de aceptación de M_1 o de M_2 respectivamente, entonces M ingresará en un estado de aceptación y se detiene. Sin embargo M debe seguir operando si una de las cintas entra en un estado de no aceptación, para ello se debe definir una transición del tipo.

$$\delta\big((p_1, q_1), (s_1, r_1)\big) = ((p_e, q_2), (s_2, S), (r_2, R_3))$$

donde p_e (o q_e según sea el caso) permite que se detenga la primera cinta y continúe la simulación de la otra ■

Teorema 7.11 Un lenguaje L es R si y sólo si L y su complemento $\bar{L}$ son RE.

Demostración parte i (Si L es R entonces L y $\bar{L}$ son RE): como L es R, por el Corolario 7.1 L es RE. Ahora, también como L es R su complemento $\bar{L}$ (Teorema 7.9) es R y nuevamente por el corolario 7.1 $\bar{L}$ es RE.

Demostración parte ii (Si L y $\bar{L}$ son RE entonces L es R): sean M_1 y M_2 dos MT que aceptan a L y $\bar{L}$ respectivamente. Se construye una MT M que simule simultáneamente a M_1 y M_2, tal como se hizo en el Teorema 7.11. Ahora, para cada cadena w sólo hay dos posibilidades que $w \in L$ o que $w \in \bar{L}$, por lo que no hay más para la máquina M sino detenerse en la cinta uno o en la cinta dos. Como M se detiene para toda entrada, L es recursivo.

Corolario 7.3 Sea L un lenguaje arbitrario, siempre se cumplirá una de las tres siguientes afirmaciones:

i. L y $\bar{L}$ son R.

ii. Ni L ni $\bar{L}$ son RE.

iii. L o $\bar{L}$ es RE (no ambos al tiempo) y el otro no es RE.

7.5 Jerarquía de Chomsky

Como se mencionó en la sección 6.3, Chomsky realizó una jerarquía entre los lenguajes regulares (tipo 3), LIC (tipo 2), R y RE (tipo 0), sin embargo no se mencionó que esta clasificación está en función de los recursos computacionales que se necesitan para reconocerlos, empero, la deducción es sencilla a partir de los teoremas de las secciones 7.3 y 7.4. Al respecto García & Martínez (2005, pp. 10-11) menciona:

> Uno de los objetivos del estudio de los lenguajes formales es asimilar la correspondencia entre lenguajes formales y problemas computacionales, esto es, cualquier lenguaje formal se puede asociar a un problema computacional. Desde este punto de vista, reconocer las cadenas de un lenguaje es equivalente a solucionar un problema. Por lo tanto, la jerarquía de Chomsky realmente lo que permite es clasificar por orden creciente el poder computacional necesario para poder resolver distintas clases de problemas. Y uno de los resultados más espectaculares de esta disciplina lo supone el hecho de que, hoy por hoy, existen problemas que no se pueden resolver aun cuando se utilice el modelo con mayor poder computacional, el correspondiente a los lenguajes de Tipo 0, que modela la capacidad computacional de cualquier computador existente. Esto es, hay problemas irresolubles, hay límites para el concepto de computación tal y como lo entendemos.

Por último a y modo de resumen, dado un lenguaje L se tiene que la jerarquía de Chomsky establece que sólo las siguientes implicaciones son válidas:

i. Si L es regular entonces L es LIC (Sección 6.3).

ii. Si L es regular entonces es L R (Teorema 7.6).

iii. Si L es regular entonces L es RE (Teorema 7.6 y Corolario 7.1).

iv. Si L es LIC entonces L es R (Teorema 7.7).

v. Si L es LIC entonces L es RE (Teorema 7.7 y Corolario 7.1).

vi. Si L es R entonces L es RE (Corolario 7.1).

7.6 La tesis Church-Turing

7.6.1 El concepto de algoritmo.

Para Hilbert resolver el Entscheidungsproblem consistía en hallar o diseñar un algoritmo que permita decidir, partiendo de una proposición y en un número finito de pasos si una conclusión es demostrable. Por tanto un algoritmo o procedimiento de decisión según Pérez & Caparrini (2003, p. 5) consiste en definir (o decidir) si cierta cantidad de símbolos de un sistema constituye o no una demostración. Este procedimiento debe recorrer todas las sucesiones finitas de símbolos del sistema y aceptar aquellas que conforman una demostración en el sistema.

Sin embargo ¿Qué se entiende por Algoritmo? Como primera instancia se recogen las palabras de Pérez & Caparrini (2003, p. 2):

> Es usual encontrar en un diccionario el término algoritmo como sinónimo de *cualquier método especial de resolución de un cierto tipo de problema*s. La palabra algoritmo debe su nombre al autor persa Abú Jáfar Mohammed ibn al-Khowarizmi, que escribió un texto en el año 825 d.C. en el que recogía una serie de procedimientos mecánicos para el álgebra […] Históricamente el primer algoritmo no trivial es debido a Euclides que entre el año 400 y 300 antes de Cristo describió un procedimiento mecánico para hallar el máximo común divisor de dos números enteros.

Más adelante Pérez & Caparrini (2003, p. 4) explican que al formalizar el concepto algoritmo se puede optar por tres vías que son fruto de las propuestas de Alan Turing, la orientación de algunos lenguajes de programación actuales y Alonso Church[28]:

i. Describir sintácticamente los dispositivos o *máquinas* cuya ejecución (definida por una semántica precisa) proporciona las *funciones computables* del modelo.
ii. Definir directamente, a través de un lenguaje de instrucciones básicas, el concepto de algoritmo como una sucesión finita de instrucciones que verifique una serie de requisitos sintácticos. A partir de esta noción, una *máquina de cálculo* será cualquier dispositivo capaz de ejecutar los algoritmos de ese modelo, y las *funciones*

[28] Es evidente que el modelo que siguió Alan Turing sigue la primera vía, en cambio la de Church con las funciones computables sigue la tercera. Finalmente la segunda vía es la orientación de algunos lenguajes de programación que surgieron en la posteridad.

computables serán aquellas que pueden ser generadas, de manera natural, por los algoritmos del mismo.

iii. Considerar un cierto conjunto de funciones distinguidas como la clase de *funciones computables* del modelo. A partir de este concepto, los *algoritmos* del modelo serán aquellos procedimientos que permitan generar, en algún sentido claramente fijado, las funciones computables; y las máquinas de cálculo serán aquellos dispositivos que puedan ejecutar los algoritmos.

Uno de los elementos centrales dentro de la tesis de Church-Turing es el concepto de algoritmo, el cual hasta el momento no se ha definido, razón por la cual no tiene sentido demostrar la tesis, sin embargo si se puede esperar que se refute algún día.

7.4.2 La Tesis de Church-Turing[29].

A mediados de la década de los treinta Alonso Church y Alan Turing formulan de manera independiente la siguiente tesis (Pérez & Caparrini, 2003, p. 12):

Toda función computable de manera efectiva puede calcularse en el modelo[30]

En otras palabras, esto quiere decir que: toda función computable puede ser calculada por una Máquina de Turing, pero si se desea, la tesis se puede expresar así también: *un problema resoluble mecánicamente es resoluble por una máquina de Turing*.

Desde el punto de vista de Turing la tesis menciona que todo algoritmo puede ser descompuesto en una secuencia de pasos muy simples y su enunciación es la siguiente (García & Martínez, 2005, p. 109):

Una función es computable si existe una Máquina de Turing
que la calcule en un número finito de pasos.

[29] La Tesis de Church-Turing recibe este nombre debido a que fue Alonso Church el primero en llegar a esta conclusión. Un año más tarde y de manera independiente Alan Turing llega a lo mismo.
[30] La expresión "función computable de manera efectiva" es equivalente a función computable de acuerdo a lo expuesto en la sección 7.3.7.

Es decir que una máquina de Turing se puede relacionar con una función que calcula y viceversa. Por tanto si se habla de una función computable la Máquina de Turing termina los cálculos en un número finito de pasos.

Más adelante García & Martínez (2005, p. 113) hacen la siguiente claridad: “En esta tesis, la noción de función computable no pone límites ni al número de pasos necesarios, ni a la cantidad de espacio de almacenamiento necesario, para desarrollar la computación” por los que se puede establecer con facilidad que estas propiedades las cumple a cabalidad una Máquina de Turing.

7.4.3 Consenso científico alrededor de la Tesis de Church-Turing.

Como se mencionó al final de la sección 7.41, no hay razones para intentar mostrar que la Tesis de Church - Turing es cierta. Al respecto Pérez & Caparrini (2003, p. 13) mencionan que es inútil buscar una demostración a esta afirmación y justifica el por qué se puede “creer” en la tesis:

> Conviene dejar bien claro que esta tesis relaciona dos conceptos de naturaleza completamente distinta: por una parte la existencia de un procedimiento mecánico que calcule una función o resuelva un problema (concepto informal) y, por otra, la existencia de una máquina de Turing que calcula una función o resuelve un problema (concepto formal). Por tanto, no tiene sentido buscar una prueba matemática de dicha tesis.
>
> Entonces ¿por qué la admitimos? Simplemente porque todos los problemas de los que se conocen soluciones consideradas como mecánicas, son también resolubles a través de máquinas de Turing (o en cualquiera de los otros dos modelos de computación).
>
> […] Ahora bien, el científico debe tener un espíritu escéptico y crítico por naturaleza. De ahí que no se puede soslayar la posibilidad de que algún día la tesis de Church-Turing pueda ser rechazada (que no refutada). De momento la llevamos aceptando desde hace cerca de setenta años.

En el mismo sentido De Castro (2004, p. 198) afirma:

> […] diseñar una MT es completamente similar a escribir un programa computacional ya que la función de transición de una MT no es otra cosa que un conjunto de instrucciones. Se establece así una conexión intuitiva directa entre máquinas de Turing y algoritmos.
>
> *[…] Tesis de Church-Turing. Todo algoritmo puede ser descrito por medio de una máquina de Turing.*
>
> En su formulación más amplia, la tesis de Church Turing abarca tanto los algoritmos que producen una salida para cada entrada como aquellos que no terminan (ingresan en bucles infinitos) para algunas entradas.
>
> Para apreciar su significado y su alcance, hay que enfatizar que la tesis de Church Turing no es un enunciado matemático susceptible de demostración, ya que involucra la noción intuitiva de algoritmo. En otras palabras, la tesis no se puede demostrar. Se podría refutar, no obstante, exhibiendo un procedimiento efectivo, que todo el mundo acepte que es un verdadero algoritmo y que no pueda ser descrito por una máquina de Turing. Pero tal refutación no se ha producido hasta fecha; de hecho, la experiencia acumulada durante décadas de investigación ha corroborado una y otra vez la tesis de Church Turing.

Finalmente García & Martínez (2005, p. 113) sustentan lo que ha venido mostrando:

> Esta es una tesis, no un teorema ni un resultado matemático: sólo establece la correspondencia entre un concepto informal (la computabilidad) y un concepto matemático (las funciones recursivas parciales). Es, por lo tanto, teóricamente posible que la Tesis de Church pueda quedar obsoleta en el futuro, si se propone un modelo alternativo para la noción de computabilidad que pueda desarrollar cálculos que no pueden realizarse mediante la Maquina de Turing. Pero no se considera probable.
>
> De hecho, hay una serie de modelos lógicos desarrollados ($\lambda-$ calculus, Sistemas de Post, Maquinas de Turing,...) y todos definen la misma clase de funciones, las funciones recursivas parciales. De entre estos modelos, hay uno más cercano al punto de vista de los programadores, el modelo RAM.

De este modo el consenso científico es claro respecto a lo que se puede esperar de la tesis, sin embargo es seguro que los avances en el tema nunca dejaran de hacerse esperar.

8. Problemas Indecidibles

La hipótesis de Church-Turing como se mencionó en el capítulo anterior da la posibilidad de diseñar Máquinas de Turing para calcular funciones parciales o, en otros términos, para aceptar lenguajes RE. Ahora, dada una función computable es posible calcular su solución, siempre y cuando esta exista, construyendo una Máquina de Turing, sin embargo no es posible garantizar que sucederá con los argumentos de entrada antes de calcularlos o inclusive cuando están siendo calculados, es decir no se puede conocer de antemano si se producirá una salida. Caso contrario, si todas las salidas están garantizadas entonces la función asociada a la MT es total y acepta un lenguaje recursivo.

Ejemplo 8.1 Un número x es maravilloso si es 1 o si puede llegarse al 1 aplicando sucesivas veces la siguiente función:

$$f(x) = \begin{cases} \dfrac{x}{2} \; si \; x \; es \; par \\ 3x + 1 \; si \; x \; es \; impar \end{cases}$$

Actualmente se conjetura que todos los números enteros son maravillosos y no existe demostración al respecto (García & Martínez, 2005, p. 125). A partir de esto se puede establecer que el dominio de la función f no esté determinado y sea una función parcial pues no es posible determinar que el proceso finalice para cualquier valor entero que se considere.

8.1 Funciones totales vs funciones parciales

Hablar de las funciones totales vs las funciones parciales es lo mismo que hablar de los lenguajes R vs los RE (sección 7.3.7). Por lo tanto, dada una función computable pueden suceder dos cosas:

i. Si la función es total, existe una MT que siempre se detiene, bien dando el resultado o arrojando un error, el error se da si los argumentos no pertenecen

al dominio de la función. Desde los lenguajes, cuando la función total es calculada para todos los argumentos del dominio, la cadena asociada pertenece al lenguaje aceptado por la MT, de lo contrario es la cadena es rechazada. El lenguaje asociado a la función total debe ser necesariamente R.

ii. Si la función es parcial, la MT asociada sólo calcula los resultados cuando los argumentos pertenecen al dominio de la función, no se puede garantizar que ocurre en caso contrario, es decir, si se calculan argumentos que no pertenecen al dominio de la función estos pueden arrojar un error o no arrojar información alguna. En este caso la MT acepta las cadenas relacionadas con los argumentos del dominio de la función, pero para las otras, las rechaza o no sabe qué hacer. El lenguaje asociado a la función total es RE.

Lo anterior sólo es una descripción más de los lenguajes que se trataron en el capítulo anterior y una ampliación de lo mencionado en la sección 7.3.7.

8.2 El concepto de problema

Definición 8.1 Un problema es un enunciado cierto o falso dependiendo de los argumentos que aparecen en su definición.

Definición 8.2 Una solución a un problema es una aplicación entre el conjunto de instancias de los argumentos del problema y el conjunto $\{cierto, falso\}$.

A partir de lo anterior se puede establecer de manera informal que un algoritmo es un conjunto de pasos cuyo objetivo es resolver un problema. Según García & Martínez (2005, p. 126) es posible identificar un algoritmo con una función:

$$f = A_1 \times A_2 \times \ldots \times A_n \longrightarrow A$$

De forma que el algoritmo obtiene un valor de salida a partir de unos valores de entrada si ese valor existe o si el problema tiene solución. La solución al problema está relacionada con la siguiente aplicación:

$$S(f) = A_1 \times A_2 \times \ldots \times A_n \times A \longrightarrow \{cierto, falso\}$$

Si se establece esta aplicación entre los argumentos $a_1, a_2, \dots, a_n, a$ y el conjunto $\{cierto, falso\}$, hay una solución al problema, la función f es total y se puede decidir si el problema es cierto o falso. En resumen:

$$S(f) = (a_1, a_2, \dots, a_n, a) = cierto \Leftrightarrow f(a_1, a_2, \dots, a_n) = a$$

En los problemas en los que los argumentos de la función f no puedan ser calculados cuando no pertenecen al dominio de la función, se tiene que el problema no es decidible, es decir, que la función definida para el conjunto de argumentos y el conjunto $\{cierto, falso\}$ es de tipo parcial, por lo que habrá argumentos que no se puedan relacionar con los elementos $cierto$ o $falso$. En resumidas cuentas se tiene:

$$Lenguaje\ recuersivo \equiv problema\ decidible$$

Entonces, si determinado problema está asociado a un lenguaje R, este es decidible, de lo contrario es indecidible. Cuando el problema es decidible existe un algoritmo para resolverlo.

8.3 Codificación binaria de las Máquinas de Turing

Es posible codificar las MT de manera binaria con finitos ceros y unos. Se conviene, a partir de este momento, que toda MT tenga un único estado inicial q_1 y un único estado final q_2. Toda MT se puede convertir en una máquina con un único estado de aceptación sin alterar el lenguaje aceptado añadiendo transiciones desde los estados de aceptación originales hasta un único estado de aceptación q_2. Estas transiciones deben ser de la forma: $\delta(q, a) = (q_2, a, S)$ para todo estado de aceptación q y $a \in \Gamma$.

Para cada MT M , el alfabeto de la cinta puede escribirse de la forma $\Gamma = \{s_1, s_2, \dots, s_m, \dots, s_p\}$, con $s_1 = \mathrm{B}$, $\Sigma = \{s_2, \dots, s_m\}$ y $s_{m+1}, \dots, s_p$ caracteres auxiliares de M. Estos símbolos se codifican como sucesiones de unos (ver Tabla 8.1):

Tabla 8.1 Codificación alfabeto de la cinta de la MT *M*

Símbolo	Codificación
$s_1 = \mathrm{B}$	1
s_2	11
s_3	111
…	…
s_m	$1 \dots 1, m\ veces$
…	…
s_p	$1 \dots 1, p\ veces$

Las cadenas de Γ^* se pueden escribir usando 0 como separador.

Ejemplo 8.2 La cadena $s_2 s_2 s_3 \mathrm{B} s_2 s_3$ ya codificada es la siguiente:

$$01101101110101101110.$$

Adviértase que en la codificación de una cadena no pueden aparecen nunca dos ceros seguidos. En términos generales la codificación de una cadena $s_{i_1} s_{i_2} \dots s_{i_k} \in \Gamma^*$ es $01^{i_1}01^{i_2}0 \dots 01^{i_k}0$, donde los exponentes de los unos indican el número de unos consecutivos.

Al igual que las cadenas, los estados de una MT $q_1 q_2 \dots q_n$ se codifican como secuencias de unos (Tabla 8.2):

Tabla 8.2 Codificación de los estados de la MT *M*

Estado	Codificación
q_1 (inicial)	1
q_2 (final)	11
…	…
q_n	$1 \dots 1, n\ veces$

Los desplazamientos R, L y S se codifican con 1, 11 y 111, respetivamente.

Ejemplo 8.3 Una transición $\delta(q_3, s_2) = (q_5, s_3, R)$ se codifica usando ceros como separadores para los estados, los símbolos del alfabeto de la cinta y la directriz de desplazamiento del siguiente modo:

$$01110110111110111010$$

En general la codificación de una transición cualquiera $\delta(q_i, s_k) = (q_j, s_l, D)$ es:

$$01^i01^k01^j01^l01^t0$$

con $t = 1,2$ ó 3, según sea el desplazamiento.

Ya conociendo como se codifican las transiciones, una MT se codifica escribiendo consecutivamente las secuencias de las codificaciones de todas sus transiciones. La codificación de una MT M generalmente es de la forma: $C_1C_2C_3 \dots C_r$ donde las C_i son las codificaciones de las transiciones de M. Debido a que el orden en el que se presentan las transiciones de una MT no es relevante, se concluye que una misma MT tiene varias codificaciones diferentes. El ejemplo 8.4 muestra la codificación de una MT dada.

Ejemplo 8.4 Dada la siguiente MT $M = (\{q_1, q_2, q_3, q_4\}, \{a, b, B\}, \{a, b\}, \delta, q_1, B,\ q_2)$ junto con la función transición δ (Tabla 8.3) el proceso para codificarla es el siguiente:

Tabla 8.3 Función de transición δ Ejemplo 8.4

Función de transición δ
$\delta(q_1, a) = (q_3, a, R)$
$\delta(q_3, b) = (q_3, b, R)$
$\delta(q_3, B) = (q_2, B, S)$
$\delta(q_4, b) = (q_4, b, R)$
$\delta(q_4, a) = (q_3, a, R)$

La codificación de la MT M empieza con el alfabeto de la cinta:

Tabla 8.4 Codificación alfabeto de la cinta de la MT *M* Ejemplo 8.4

Símbolo	Codificación
B	1
a	11
b	111

La codificación de estados de M es (Tabla 8.5):

Tabla 8.5 Codificación de los estados de la MT M Ejemplo 8.4

Estado	Codificación
q_1 (inicial)	1
q_2 (Final)	11
q_3	111
q_4	1111

Ahora, la codificación de cada transición de la MT M es la siguiente (Tabla 8.6):

Tabla 8.6 codificación de cada transición de la MT M Ejemplo 8.4

Función de transición	Codificación
$\delta(q_1, a) = (q_3, a, R)$	010110111011010
$\delta(q_3, b) = (q_3, b, R)$	0111011101110111010
$\delta(q_3, B) = (q_2, B, S)$	0111010110101110
$\delta(q_4, b) = (q_4, b, R)$	011110111011110111010
$\delta(q_4, a) = (q_3, a, R)$	011110110111011010

Luego la MT M se puede codificar como:

$$0101101110110100111011101110111010011101011010111001111011101111011101001111011011101 1010$$

Finalmente se podría reducir la codificación a:

$$01^1 01^2 01^3 01^2 01^1 001^3 01^3 01^3 01^3 01^1 001^3 01^1 01^2 01^1 01^3 001^4 01^3 01^4 01^3 01^1 001^4 01^2 01^3 01^2 01^1 0$$

No todas las secuencias binarias representan una MT, por ejemplo, en la codificación de una MT no pueden aparecer tres ceros seguidos y tampoco pueden comenzar por uno. Dado el caso, si una cadena binaria no representa una MT, se supondrá que codifica la MT con un solo estado y sin transiciones y que acepta el lenguaje ∅. De este modo se puede hablar de la cadena binaria *i*-ésima y de la *i*-ésima MT. Sin embargo, debe tenerse en cuenta que cada MT $M_1, M_2, \ldots$ aparece varias veces ya que al cambiar el orden de las transiciones se obtiene una codificación diferente. Las MT que aceptan el lenguaje vacío aparecen infinitas veces.

Debido a que una máquina de Turing se puede relacionar con una función calculable y como una MT se puede describir mediante una cadena finita de caracteres es innegable que el número de máquinas de Turing (y el número de funciones computables) es infinito pero numerable (Pérez y Caparrini p. 109).

Teorema 8.1 El conjunto de Máquinas de Turing y de las funciones computables se pueden ordenar en orden lexicográfico.

Reformulación: El conjunto de cadenas que codifican las MT sobre $\Sigma = \{0,1\}$ es un lenguaje R.

Demostración: puesto que las MT se codifican a partir del alfabeto $\Sigma = \{0,1\}$ se tiene que toda cadena del lenguaje Σ^* es la codificación de una MT (incluyendo las que aceptan el lenguaje vacío). Σ^* es un lenguaje regular por el Teorema 6.8, también por el Teorema 7.5 Σ^* es un lenguaje R, se tiene entonces que el conjunto de las Máquinas de Turing y las funciones computables se pueden ordenar en orden lexicográfico por el Teorema 7.4 ■

Corolario 8.1 El conjunto de cadenas que codifican las MT sobre $\Sigma = \{0,1\}$ es enumerable.

8.4 Máquina de Turing Universal

La Máquina de Turing Universal M_u simula el comportamiento de todas las MT sobre Σ. M_u admite pares de la forma (M, w), siendo M la codificación de una MT y w la codificación de una cadena de entrada para M. El par (M, w) se presenta como una cadena en la forma $M0w$ en binario. Puesto que el código de M termina en cero y el de w comienza con cero, la cadena $M0w$ sólo contiene en una ocasión tres ceros consecutivos.

Definición 8.3 La Máquina Universal de Turing M_u, es una MT con una sola cinta cuyo alfabeto es $\{0,1,\mathrm{B}\}$ y que acepta el lenguaje L_u

Definición 8.4 El lenguaje aceptado por M_u es el lenguaje universal denominado L_u y cuyo conjunto de cadenas es:

$$L_u = \{M0w: M\ acepta\ la\ cadena\ w \in \Sigma^*\}$$

Teorema 8.2 L_u es un lenguaje recursivamente enumerable.

Demostración: se construye una MT M' con tres cintas cuyo alfabeto es $\{0,1,\mathrm{B}\}$. Las entradas en cada una de las cintas es la siguiente:

i. La primera cinta contiene el código de una MT M determinada.
ii. La segunda cinta contiene inicialmente el código de una entrada w para M.
iii. La tercera cinta se usa para almacenar el estado actual de M,

Con la entrada $M0w$ la máquina M' procede así:

i. Se colocan los códigos de M y w en la primera y segunda cintas respectivamente.
ii. La cadena 1 que representa el estado inicial q_1 se coloca en la tercera cinta. La UC escanea inicialmente el primer símbolo de cada cadena binaria, en cada una de las tres cintas.
iii. La UC escaneando la primera cinta con el primer visor examina el código de M y determina si representa una MT válida. En caso negativo M' se detiene sin aceptar.
iv. Si el paso anterior es caso afirmativo, M' utiliza la información de la segunda y tercera cintas para buscar en la cinta 1 la transición que sea aplicable. Si M' encuentra la transición para el símbolo leído en la cinta 2 por M, cambia el estado señalándolo en la cinta 3. La simulación continúa de esta forma siempre y cuando haya transiciones aplicables. Si al procesar la cadena w, M' se detiene en el único estado de aceptación de M, entonces la cadena w será aceptada por M y M'.
v. Puede suceder que M' no encuentre una transición aplicable o que se detenga en un estado de no aceptación. En este caso la cadena w no es aceptada por M y M'.

Se tiene que M' acepta la entrada $M0w$ si y sólo si M acepta a w. El lenguaje aceptado por la MT M' es L_u. Por el Teorema 7.2 y 7.1 se está garantizando la existencia de M_u que también acepta el lenguaje L_u ■

8.5 Un lenguaje no recursivamente enumerable

Se presenta a continuación un lenguaje que no es RE. En él se destaca el método de la diagonal de Cantor[31]. La construcción del lenguaje diagonal L_d es el siguiente:

Se construye una tabla de doble entrada (Tabla 8.7). La primera fila representa las MT M_j en orden lexicográfico y la primera columna las cadenas $w_i \in \Sigma^*$ ordenadas de la misma manera[32]. La entrada para el resto de celdas de la tabla corresponde a la siguiente regla: si $w_i \in L(M_j)$ entonces el elemento $(i,j) = 0$, si $w_m \notin L(M_j)$ entonces $(i,j) = 1$.

Tabla 8.7 Construcción del lenguaje diagonal

	M_1	M_2	M_3	...	M_m
w_1	**1**	1	1	...	0
w_2	1	**0**	0	...	1
w_3	1	1	**1**	...	0
⋮	⋮	⋮	⋮	⋱	⋮
w_n	1	1	1	...	**0**

Definición 8.5 El lenguaje diagonal representado por L_d y corresponde al conjunto:

$$L_d = \{w_i : w_i \; no \; es \; aceptada \; por \; M_i\}$$

Teorema 8.3 El lenguaje diagonal L_d no es RE, es decir que no es aceptado por ninguna MT.

Demostración: supóngase que L_d es RE. Por la definición 7.7 existe una MT M_z que acepta a L_d por tanto $L_d = L(M_z)$ para algún z, luego:

- Si $w_z \in L_d$ entonces por definición de L_d, w_z no es aceptada por M_z, entonces $w_z \notin L(M_z)$ y por sustitución $w_z \notin L_d$.

[31] Cantor uso su diagonalización para demostrar que el conjunto de los números reales no es numerable.
[32] La disposición de las cadenas y de las MT en orden lexicográfico sobre $\Sigma = \{0,1\}$ en la Tabla 8.7 garantiza que todas se encuentren allí por el Teorema 8.1.

- Si $w_z \notin L_d$ entonces por sustitución $w_z \notin L(M_z)$, entonces por la Definición 8.5 de L_d, w_z es aceptada por M_z, entonces $w_z \in L(M_z)$ y por sustitución $w_z \in L_d$.

Por tanto se genera la contradicción $w_z \in L_d \leftrightarrow w_z \notin L_d$, por lo que no existe una MT M_z que acepte al lenguaje L_d ■

Teorema 8.4 El lenguaje L_u es RE pero no es R.

Demostración: supóngase que L_u es R, por tanto existe una MT M' que procesa todas las cadenas $M0w$ y se detiene en un estado de aceptación (si $w \in L(M)$) o de rechazo (si $w \notin L(M)$), luego si esto sucede, existe en particular una MT M'' que acepta a L_d de esta manera: dada una cadena $w \in \Sigma^*$, M'' es una MT que puede enumerar las cadenas de Σ^* hasta encontrar $w_k = w$, luego M'' simula a M' con la entrada $M_k 0 w_k$, y luego decide si M_k acepta a w_k. Si $w_k \in L(M_k)$, M'' acepta y si $w_k \notin L(M_k)$, M'' rechaza. De este modo es posible construir una MT M''' que haga lo contrario de M'', es decir, si $w_k \notin L(M_k)$, M''' acepta y si $w_k \in L(M_k)$, M''' rechaza. Se tiene que M''' acepta el lenguaje L_d, de lo que se concluye que L_d es un lenguaje RE y esto contradice el Teorema 8.3 ■

A partir de la demostración del Teorema 8.4 se deduce que no se puede construir una máquina que simule a una máquina para preguntarle si acepta una cadena antes de ser procesada. Como conclusión el siguiente problema resulta ser indecidible:

Dada una Máquina de Turing M y una cadena w ¿acepta M la cadena w?

8.6 El problema de la Parada

El problema de la parada está relacionado con la existencia de una MT que determina si otra MT se para o no. En inglés este popular problema recibe el nombre de *Halting problem* y se puede enunciar de la siguiente manera:

Dada una MT M cualquiera, sobre un alfabeto de la cinta Σ y una cadena $w \in \Sigma^*$

¿Se detiene M al procesar la cadena w?

Teorema 8.2 El problema de la parada es indecidible.

Demostración: Suponemos que existe la MT que decide a priori si M_m acepta a w_n. Se construye la MT M_p de tal modo que acepte cadenas de la forma $M_i 0 w_j$ si $w_j \in L(M_i)$ y si $w_j \notin L(M_i)$ las rechace. Luego de que M_p procese cada cadena se establece que actué así:

$$M_p(M_i 0 w_j) = \begin{cases} escribe\ 0 = falso\ , si\ w_j\ se\ va\ a\ un\ bucle\ infinito\ con\ M_i\ o\ M_i\ no\ para. \\ escribe\ 1 = cierto, si\ w_j\ es\ aceptada\ o\ rechazada\ por\ M_i\ o\ M_i\ se\ para. \end{cases}$$

Se tiene entonces que M_p escribirá en una de sus cintas como resultado 0 ó 1 según sea el caso[33]. Ahora, se construye una MT M_z de tal modo que cuando reciba cadenas de la forma $M_p 0 M_i 0 w_j$ realice los dos siguientes movimientos de escritura[34]:

$$M_z(M_p 0 M_i 0 w_j) = \begin{cases} escribe\ 0, si\ M_p\ escribió\ 0 \\ escribe\ 1, si\ M_p\ escribió\ 1 \end{cases}$$

En particular se tiene que M_z al recibir $M_p 0 M_s 0 w_s$ sucede:

i. Si M_z escribe 0, entonces debe suceder que M_p al recibir la cadena $M_s 0 w_s$ escribe 0, a su vez si esto ocurre, la consecuencia inmediata es que M_s no parara al recibir la cadena es w_s.
ii. Si M_z escribe 1, entonces debe suceder que M_p al recibir la cadena $M_s 0 w_s$ escribe 1, a su vez si esto ocurre, la consecuencia inmediata es que M_s para al recibir la cadena es w_s.

En otros términos:

$$M_z(M_p 0 M_s 0 w_s) = \begin{cases} escribe\ 0, si\ M_p\ escribió\ 0, es\ decir, M_s\ no\ para\ al\ recibir\ w_s \\ escribe\ 1, si\ M_p\ escribió\ 1, es\ decir, M_s\ acepta\ o\ rechaza\ a\ w_s \end{cases}$$

Pero ¿qué sucede si M_z recibe su propio código? Para ello se supone la existencia de una cadena w_z tal que $w_z = M_z$. Si M_z recibe la cadena $M_p 0 M_z 0 w_z = M_p 0 M_z 0 M_z$ se tienen dos casos:

[33] Téngase en cuenta que ninguna cadena que procese M_p no quedará en un bucle infinito.
[34] M_z también tiene las mismas características señaladas en M_p, es decir que M_z siempre para.

i. Si M_z escribe 0, entonces debe suceder que M_p al recibir la cadena $M_z 0 M_z$ escribe 0, a su vez si esto ocurre, la consecuencia inmediata es que M_z no parara al recibir la cadena $w_z = M_z$, lo que constituye una contradicción. Se tiene que M_z no es una MT que acepte un lenguaje R tal como se había construido.

ii. Si M_z escribe 1, entonces debe suceder que M_p al recibir la cadena $M_z 0 M_z$ escribe 1, a su vez si esto ocurre, M_z para al recibir la cadena $w_z = M_z$. Entonces M_z sólo acepta y rechaza cadenas, es decir que está asociada a un lenguaje R, pero del mismo modo que la demostración del Teorema 8.4 es posible aquí en este segundo caso es posible construir una MT que a partir de M_z acepte el lenguaje L_d por lo que este caso no es posible.

De este modo se establece que el problema de la parada no tiene solución ■

8.7 El Entscheidungsproblem no tiene solución

Para aproximar la demostración del problema de la parada a la solución negativa del Entscheidungsproblem, se parte del supuesto que el problema de la decisión es decidible y se establecen las siguientes condiciones iniciales:

i. El alfabeto Σ es un conjunto finito y está conformado por todos los símbolos necesarios para trabajar con la lógica de predicados o de primer orden y el elemento "0". Vale la pena aclarar que el número de variables también es finito.

ii. Cada teorema o conclusión w_i del sistema se puede escribir en términos de los elementos de Σ o como cadenas de Σ^*.

iii. Es razonable suponer que cada teorema del sistema está asociado a un procedimiento que lo afirma, lo desmiente o no dice algo sobre él.

iv. Un procedimiento en la tercera condición inicial es un algoritmo.

v. Por las condiciones iniciales iii. y iv. cada teorema está asociado a un algoritmo.

vi. Por la tesis de Church-Turing cada teorema está asociado a una MT.

vii. Cada MT está en la capacidad de decidir si acepta o rechaza una cadena w_i, esto quiere decir que si el procedimiento o algoritmo que la MT contiene demuestra la proposición w_i, entonces la MT acepta a w_i, ahora, si el algoritmo de la MT no demuestra a w_i, entonces la MT rechaza w_i. Sin embargo estos dos casos no son forzosos ya que existe un tercero en el que la cadena w_i se va a un bucle infinito al ser leída por una MT.

viii. Cada MT es posible codificarla en una cadena $M_j \in \Sigma^*$.

Supóngase que existe un algoritmo, es decir una MT M_p, de modo que al recibir la cadena $M_j 0 w_i$ decida si M_j constituye una demostración de w_i. Como aún se sigue suponiendo que el problema de la decisión de Hilbert es decidible entonces se establece que M_p actué de la misma manera como lo hace en la demostración del problema de la parada, es decir:

$$M_p(M_i 0 w_j) = \begin{cases} escribe\ 0, si\ w_j\ se\ va\ a\ un\ bucle\ infinito\ con\ M_i\ o\ M_i\ no\ para. \\ escribe\ 1, si\ w_j\ es\ aceptada\ o\ rechazada\ por\ M_i\ o\ M_i\ se\ para. \end{cases}$$

Del mismo modo se construye una MT M_z que cuando reciba cadenas de la forma $M_p 0 M_i 0 w_j$ realice los dos siguientes movimientos de escritura:

$$M_z(M_p 0 M_i 0 w_j) = \begin{cases} escribe\ 0, si\ M_p\ escribió\ 0 \\ escribe\ 1, si\ M_p\ escribió\ 1 \end{cases}$$

Siguiendo exactamente el mismo procedimiento de la demostración de la indecibilidad del problema de la parada se llega a que el Entscheidungsproblem no tiene solución con el modelo establecido por Alan Turing ■

8.8 Consideraciones finales

Con base en la tesis de Church-Turing es posible considerar que un sistema axiomático es decidible cuando existe un algoritmo con el que se puede establecer si una sucesión de símbolos (fórmula o cadena) es lógicamente válida con los argumentos de la sección 8.2. Por ejemplo, lógica proposicional provee la forma de determinar si

una fórmula es lógicamente válida por medio de las tablas de verdad y la evaluación de tautologías. Este resultado se relaciona con el teorema de la completitud de Gödel, donde se menciona la posibilidad de escoger a partir de un lenguaje de primer orden algunas fórmulas como axiomas y algunas reglas de inferencia de modo tal que todas las formulas sean lógicamente válidas, demostrables y decidibles.

Otro ejemplo de un sistema decidible lo constituye la lógica de predicados monádicos, pues este es decidible (Cadevall, 1976, p. 459)[35]. Respecto a la relación de un sistema lógico y un lenguaje determinado Cadevall más adelante menciona: ”Un cálculo es decidible si y solamente si el conjunto de sus tesis (enfoque sintáctico) o el conjunto de sus fórmulas válidas (enfoque semántico) es recursivo”.
En este sentido y de acuerdo a la sección 8.7, la lógica de primer orden o de predicados es indecidible junto la aritmética de Peano y cualquier sistema axiomático propuesto basado en estos.

*****

Según Hernández & Morado (2007, p. 313), Hilbert expresó en cuatro puntos las cosas que se de deseaban de un procedimiento efectivo, a saber: (1) un cálculo o procedimiento es un conjunto de instrucciones a ser ejecutadas para resolver un problema; (2) debe reducirse a reglas para manipular fórmulas de un lenguaje formal adecuado ; (3) debe garantizar la solución del problema pertinente por medio de un número finito de pasos; (4) debe ser posible acotar de antemano cuantos pasos llevará encontrar la solución. Se comentará sobre estos requerimientos a continuación.

Para Alan Turing no fue posible encontrar la manera de definir *procedimiento efectivo* según el requerimiento de Hilbert, sin embargo la máquina propuesta por Turing se acerca a lo que intuitivamente se entiende por algoritmo. Las razones por las cuales una MT no es equivalente a un *procedimiento efectivo* son las siguientes:

i. Una MT no garantiza la solución en un número finito de pasos cuando, por ejemplo, problema está asociado a un lenguaje recursivamente enumerable y,

[35] Para más información diríjase a Cadevall (1976, pp. 455-484).

ii. de acuerdo a la sección siete y ocho de este trabajo, una Máquina de Turing no garantiza a priori los la cantidad de pasos computacionales que son necesarios para aceptar una cadena; la imposibilidad de la existencia de una máquina con estas características se deduce de la demostración de la indecibilidad del problema de la parada, pues si no existe una MT que diga si otra MT se detiene o para al procesar una cadena determinada, mucho menos se le puede exigir que diga en cuantos pasos lo hace.

Por lo que se concluye que el tercer y cuarto requerimiento de Hilbert no son satisfechas del todo por las MT.

Respecto al primer requerimiento, dado un problema decidible y una MT asociada, esta ha sido vista, según la tesis de Church-Turing, como un conjunto de instrucciones gracias a las funciones de transición. El segundo requerimiento de Hilbert hace referencia a que la resolución de un problema es un conjunto de transformaciones sintácticas, por lo que se puede afirmar que los problemas son representables según el alfabeto sobre el que trabaja una determinada MT (Hernández & Morado, 2007, p. 316,317). Se puede concluir de manera superficial que el primer y segundo requerimiento de Hilbert son posiblemente satisfechos por las MT cuando están ligados a problemas decidibles.

Conclusiones

- La demostración de la indecibilidad del Problema de la Parada, por medio de las Máquinas de Turing, permite establecer una solución negativa al Entscheidungsproblem. Sin embargo, llegar a esta conclusión implica admitir que la noción de algoritmo es equivalente al modelo propuesto por Alan Turing, es decir que se acepta la tesis de Church-Turing.
- La indecibilidad de la lógica de primer orden está relacionada con la estructura misma del sistema. Por ejemplo, la Aritmética de Peano a pesar de trabajar con el conjunto numerable de los números naturales y con finitos símbolos implica la existencia de funciones no computables relacionadas con la no enumerabilidad.
- El poder computacional de las Máquinas de Turing es limitado; es por ello que a pesar de que hay una conexión intuitiva entre la noción de algoritmo y la Máquina de Turing no se descarta la posibilidad de que la tesis de Church-Turing pueda ser un día refutada.
- Una solución para el Entscheidungsproblem según Hilbert debe cumplir con los presupuestos establecidos por la noción de procedimiento efectivo. Se encontró, a partir de la definición de una Máquina de Turing, los requerimientos de Hilbert son satisfechos de manera parcial pues no es posible que una MT establezca a priori los pasos necesarios para computar una cadena.
- El estudio de los lenguajes regulares y la teoría de Autómatas, previo al de las máquinas de Turing, permite conocer con mayor facilidad las propiedades computacionales de los lenguajes aceptados por una Máquina de Turing.
- La diferencia significativa entre las MT y los autómatas es el tipo de movimientos del cabezal, además de los dominios y rangos de las funciones de transición.
- El lenguaje aceptado por un autómata es recursivamente enumerable, pues cualquier autómata puede ser simulado usando las máquinas de Turing. Empero, no toda Máquina de Turing se puede representar con autómatas.

Anexo 1: JFLAP Versión 7.0

JFLAP (del inglés *Java Formal Languaje and Automata Package*) es un software libre que permite experimentar con lenguajes formales y autómatas, en particular esta herramienta permite simular autómatas finitos, autómatas de pila y Máquinas de Turing, entre otras máquinas más. JFLAP puede ser visto también como un paquete de herramientas gráficas que pueden ser utilizados como una ayuda en el aprendizaje de los conceptos básicos de Teoría de Autómatas y Lenguajes que son vistos en un curso en cualquier curso introductorio de Teoría de la Computación.

JFLAP brinda a sus usuarios en la página web www.jflap.org una completa información respecto a la aplicación misma, sus orígenes, tutoriales, descargas, estadísticas, ejercicios, entre otros.

Anexo 2: Simulaciones en JFLAP

Con el fin de familiarizar al lector con el estudio de los distintos tipos de autómatas y las Máquinas de Turing, se presenta en un CD Anexo la simulación de los autómatas y MT ejemplificados a lo largo de este trabajo realizados en JFLAP.

El CD anexo contiene dos carpetas, la primera contiene el archivo que permite descargar y ejecutar la aplicación y la segunda contiene los archivos .jff que contienen los ejemplos de los autómatas y MT y que deben se abiertos desde JFLAP.

Bibliografía

Apostol, T. (2001). *Calculus* (Segunda edición, Vols. 1-2, Vol. 2). Barcelona: Reverté.

Beuchot, M. (1986). La conceptografía y la lógica formal de Frege. *Revista Elementos, Ciencia y Cultura*, *2*(9), 70-75.

Beuchot, M. (2008). *Introducción a la lógica*. México, D.F: UNAM.

Beuchot, M. (2011). *Manual de Filosofía*. México, D.F: Ediciones Paulinas.

Boole, G. (1854). *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. New York: Dover Publications.

Cadevall, S. (1976). Decibilidad de la Lógica de Predicados Monádicos por el Método de la Recusación. *Teorema: Revista Internacional de Filosofía*, *6*(3-4), 455-484.

Davis, M. (2002). *La computadora universal. De Leibniz a Turing*. Barcelona: Debate.

De Castro, R. (2004). *Teoría de la computación. Lenguajes, autómatas, gramáticas* (Primera edición). Bogotá: Universidad Nacional de Colombia.

Gallardo, D., Arques, P., & Lesta, I. (2003). *Introducción a la teoría de la computabilidad* (Segunda Edición). Alicante: Publicaciones Universidad de Alicante.

García, L., & Martínez, G. (2005). *Apuntes de Teoría de Autómatas y lenguajes formales*. Recuperado a partir de http://repositori.uji.es/xmlui/handle/10234/5995?locale-attribute=es

Hernández, F., & Morado, R. (2007). Hilbert, Turing y la Noción de Procedimiento Efectivo. En *Sociedad y ciencia* (pp. 313-320). México D.F: Siglo XXI editores.

Holcombe, M. (1982). *Algebraic Automata Theory*. Cambridge University Press.

Martínez, P. (1985). El origen histórico de la lógica matemática: Boole. *Thémata: Revista de filosofía*, (2), 87 - 98.

Meléndez, R. (1992). Observaciones sobre la demostración de Gödel. En P. Gómez & C. Gómez, *Sistemas Formales, informalmente ¿Por qué intentaron formalizar a la matemática si era tan buena muchacha?* (Segunda edición, pp. 109-111). Bogotá: Una empresa docente/Universidad de los Andes. Recuperado a partir de http://funes.uniandes.edu.co/668/1/Gomez1999Sistemas.pdf

Muñoz, J. (2002). *Introducción a la teoría de conjuntos* (Cuarta edición). Bogotá: Universidad Nacional de Colombia.

Navarrete, I., Cárdenas, M., Sánchez, D., Botía, J., Marín, R., & Martínez, R. (2003). *Teoría de autómatas y lenguajes formales*. Universidad de Murcia. Recuperado a partir de http://tux.uis.edu.co/lenguajes/doc/Murcia.pdf

Pérez, J., & Caparrini, F. (2003). *Máquinas moleculares basadas en ADN*. Universidad de Sevilla.

Wise, H. (1996). *Breaking the Code*. Documental, biografía, British Broadcasting Corporation (BBC). Recuperado a partir de https://www.youtube.com/watch?v=S23yie-779k