Introducción a los límites teóricos de la computación

Juan A. Ojeda
Centro de ciencias de la complejidad
Universidad Nacional Autónoma de México
Ciudad Universitaria, D.F., México,
Universidad de Guadalajara
Guadalajara, Jalisco, México.
teatrogdl@gmail.com

Septiembre 4 del 2011

Contenido

Resumen

En este artículo se presenta una breve introducción a la teoría de la computación y los sistemas formales con la intención de entender los conceptos básico de computabilidad, complejidad y teoría de autómatas, para

la comprensión de la teoría de la computación no convencional, como caso de estudio de los autómatas celulares. Partiendo de un marco histórico de los origenes de la computación se introduce al campo de la computabilidad con el fin de entender qué se puede solucionar por medio de un sistema formal como la computadora. Se plantean los problemas que dieron origen a la computación y los que se han ido desarrollando conforme se incrementan los recursos computacionales, sin que se encuentre aún una solución para ellos por su alto grado de complejidad y de demanda de recursos y se desarrolla una sección de la teoría de autómatas que presenta diferentes modelos que se utilizan en la computación y sus diferentes, funcionamientos, propiedades y capacidades para computar.

1 Introducción

La computación, sus bases y fundamentos, tiene sus orígenes en investigaciones científicas aparentemente no relacionadas con la computación e incluso antes de que esta existiera. Estas investigaciones datan de principios del siglo XX realizadas por matemáticos del área de la lógica y los sistemas axiomáticos. Debido a esto se presentan aquí algunos conceptos que ayudarán al entendimiento de estas áreas [1]:

- *Axioma*: enunciado básico que describe las propiedades fundamentales del sistema que se estudia.
- *Teoremas*: enunciados adicionales, derivados de los axiomas, aplicando secuencias finitas de reglas de inferencia.
- *Reglas de inferencia*: son reglas diseñadas para que los teoremas que se deriven sean enunciados que constituyen una consecuencia lógica de los axiomas.

Para ejemplificar, una regla de inferencia muy conocida es *modus ponens* la cual establece la siguiente propiedad:

Si X $\Longrightarrow$ Y
y X
$\therefore$ Y

En la anterior deducción nuestro axioma es "si está oscuro (X), entonces, es de noche (Y)", nuestra premisa X es "está oscuro", por lo tanto, "es de noche" es nuestro teorema Y.

La estructura del presente artículo está dividido de la siguiente forma: se empieza en la sección 1 definiendo conceptos básicos de la lógica y dando una breve introducción a las secciones del artículo; en la sección 2 se comenta un repaso histórico por diferentes acontecimientos que fueron dando origen a la teoría de la computación; la sección 3 hace una introducción al área de de la *computabilidad*, la jerarquía de funciones computables y la construcción de un lenguaje básico para computar; al llegar a la sección 4 se mostrará una clasificación de

los problemas y ciertas características que permiten la clasificación, además se verán ejemplos que aterricen la clasificación y una breve introducción a las propuestas de solución para los problemas intratables; En la sección 5 se explican las características y funcionamiento de las 3 máquinas de mayor relevancía en la teoría de autómatas, en la sección también se relacionan los autómatas con los lenguajes que pueden procesar; Por último se comentan algunas conclusiones y agradecimientos en las dos últimas secciones del documento.

2 Antecedentes

En el siglo XIX diversos lógicos de todo el mundo intentaban aclarar el término *Demostración* en el contexto de la teoría de los números debido a las paradojas que en ella se presentaban. Hubo diversas contribuciones, Giuseppe Peano usó el razonamiento formal en el estudio de conjuntos y números. David Hilbert, por su parte, trabajó en formalizaciones más estrictas que las de Euclides. Pero, sin duda, la obra inmensa y laboriosa de Bertrand Russell y Alfred North Whitehead, *Principia Mathematica* publicada entre 1910 y 1913, fue el mayor esfuerzo por dotar de completitud y eliminar toda paradoja de la teoría de los números derivando toda la matemática por medio de la lógica, incluyendo sus contradicciones. Pero, ¿Era este esfuerzo suficiente, se había logrado eliminar toda contradicción del sistema y los métodos eran congruentes consigo mismos?

Hilbert presentó el siguiente reto: demostrar rigurosamente que el sistema definido en los *Principia Mathematica* es no sólo coherente (a salvo de contradicciones), sino también completo (o sea que todo teorema válido de la teoría de los números pueda desarrollarce dentro del sistema formal de los *Principia Mathematica*) [2].

En 1931, Kurt Gödel hizo un descubrimiento transcendental, explorando el razonamiento matemático con el razonamiento matemático. Como fruto de ello obtuvo el Teorema de Incompletitud. El teorema indica que para cualquier sistema axiomático válido de teoría de los números, existen enunciados válidos (teoremas) dentro de este sistema que no pueden ser *demostrados* a partir del sistema mismo (*incompletitud*). Este logro es consecuencía del isomorfismo que Gödel encontró en las proposiciones lingüísticas auto-referenciales y las proposiciones matemáticas auto-referenciales. Estas proposiciones causan paradojas en el lenguaje, como es el caso de la *paradoja de Epiménides* la cual dice: "Todos los cretences son mentirosos" siendo Epiménides un Cretence. De una forma más clara: "Esta afirmación es falsa", una paradoja evidente. Pero no es tan evidente como una proposición relativa a los número puede hablar *acerca de* sí misma. Para demostrar la aseveración, Gödel creó la *numeración de Gödel* para que los números cumpliera la función de las proposiciones y así que una proposición de teoría de los números pudiera hablar *acerca de* una proposición de teoría de los números (inclusive *acerca de* sí misma).

El teorema de Gödel se aplica a cualquier sistema axiomático que se diga completo y coherente e implica que todo sistema que demuestre la teoría de los

números no es coherente. A su vez si el sistema es coherente no es capaz de producir todas las verdades relativas a la teoría de los números. El teorema tuvo implicaciones muy importantes en los procesos computacionales. Debido a la incompletitud, el fracaso de los sistemas axiomáticos, se presentó la duda de ¿Qué se puede resolver con los medios axiomáticos? Se crearon diversos métodos para conocer la respuesta, todos ellos *equivalentes* y conforman el núcleo de la teoría de la computación, entre estos métodos se encuentra la máquina teórica desarrolladas por Alan Turing, la *máquina de Turing*, una máquina capaz de recibir cadenas de entrada, realizar una serie de instrucciones sobre estas y arrojar una respuesta (salida), vaya, una computadora muy elemental pero en el núcleo realiza lo mismo que una computadora actual. Otro método fue el sistema de lógica y proposiciones de Church denominado *calculabilidad efectiva*. Emile Post trabajó en la construcción de la *máquina de Post* para demostrar el *problema de correspondencia de Post*. Los anteriores son ejemplos de una *computación*, por su funcionamiento, con ellos se demostró que hay problemas que se pueden resolver mediante *algoritmos* y otros en los que la máquina axiomática, como la computadora, no es suficiente [4]. Entenderemos a un *algoritmo* como una secuencia de instrucciones finitas y no ambiguas que resuelven un problema, y una *computación* como la aplicación de un conjunto de datos de entrada a un algoritmo, obteniendo un conjunto de datos de salida, resultado del proceso [3].

El campo de la teoría de la computación se ha desarrollado con la teoría de la computabilidad, teoría de la complejidad y teoría de lenguajes formales y autómatas.

3 Computabilidad

La computabilidad busca identificar aquellas *funciones computables*. Computable o calculable se refiere a la posibilidad de construir un algoritmo que permita el cálculo de una función sin importar el lenguaje o la implementación del algoritmo.

Para este estudio se codifica cualquier dato como una cadena de unos y ceros, con ello se entiende que las entradas que reciba una función computable serán tuplas de *enteros no negativos* codificadas en cadenas de unos y ceros (nótese como empieza a haber una relación más notable con la teoría de los números). Para tomar en cuenta funciones cuyo dominio no es continuo (por ejemplo, la división no es continua cuando el denominador es cero) se utiliza el término de *función parcial*. Una función parcial denota a una función de X la cual, su dominio es un subconjunto de X. En cambio, cuando el dominio de la función es todo el conjunto X, se le llamará una *función total*.

Existen 3 *funciones iniciales* que son la forma más básica que compone a las funciones computables:

- La *función cero*, representada por el símblo ζ.
- La *función sucesor*, representada por el símbolo σ.

- La *función de proyección*, representada por el símbolo π.

Estas funciones forman la base de la jerarquía de las funciones recursivas.

La mayoria de las funciones básicas que realiza una computadora actual están basadas en las *funciones recursivas primitivas*, todas estas son *funciones totales* [1]. La clase de las *funciones recursivas primitivas*, consiste en todas aquellas funciones que pueden construirse a partir de las *funciones iniciales* aplicando un número finito de combinaciones, composiciones y recursividades primitivas. Esta clase abarca la mayoría de las funciones totales, pero no todas. Por ejemplo, una función recursiva *total* que no es primitiva recursiva (que no está compuesta por funciones iniciales) es la función de Ackermann. La clase de las funciones recursivas primitivas, no abarca toda la colección de funciones computables, ya que existen funciones computables que no son *totales* como la función de Ackermann.

La clase de las *funciones totales* computables se conoce como clase de las funciones μ-recursivas y abarca funciones iniciales, recursivas primitivas y totales computables [1].

La siguiente clase en la jerarquía de computabilidad, es la clase de las *funciones computables estrictamente parciales*, para su estudio introduciremos el concepto de minimalización, el cual está relacionado con la inducción matemática. El método consiste en dividir la función recursiva añadiendo una cláusula extra que retorna el valor más pequeño de la función, como sucede en la inducción, para poder resolver la función con el conjunto de elementos más pequeños. Esta clase se contruye a partir de las funciones iniciales y un número finito de combinaciones, composiciones, recursividades primitivas y minimalizaciones, esto permite que la clase contenga todas las *funciones parciales computables*, como una *máquina de Turing* posee el poder de realizar cualquier *computación*.

La clase de las *funciones recursivas parciales*, figura 1, representa la tesis de Church, que hasta hoy en día tiene validez, ya que nadie ha encontrado una *función parcial computable* que no sea una *función recursiva parcial*. Pero no sólo esto da fuerza y validez a la tesis de Church, hay otro aspecto importante que señala la validez de la tesis por su equivalencia a la tesis de Turing, a continuación se muestran los dos teoremas válidos que lo comprueban.

- Todo proceso computacional realizado por una máquina de Turing es el cálculo de una función recursiva parcial.

- Toda función recursiva parcial es computable por una máquina de Turing.

La conjunción de ambos teoremas nos muestra que ambas tesis son iguales y concluimos el mismo límite del *alcance de los procesos computacionales*, desde dos enfoques distintos el operacional y el funcional, este límite es el conjunto de las funciones recursivas parciales.

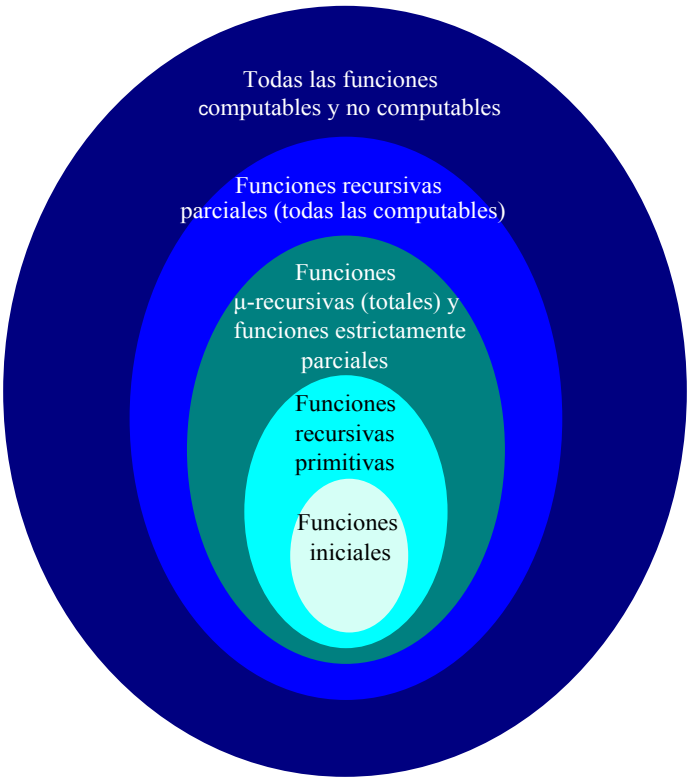

Figura 1: Diagrama de Venn donde se muestra la jerarquía final de los problemas computables con el enfoque de funciones recursivas.

3.1 Lenguaje básico de programación

Una vez que tenemos definidos los alcances de los algoritmos computacionales podemos indagar en los requerimientos mínimos [1] de un lenguaje de programación capaz de implementar la solución a cualquier *función recursiva parcial*, ya sea total o estrictamente parcial. El grupo de requerimientos se compone de:

- El conjunto de los enteros no negativos $\mathbb{N}^+$ será el tipo de dato que se utilizará.
- Dos enunciados de asignación:

  incr nombre; decr nombre;

  El primero incrementa en una unidad la variable *nombre* y el siguiente la decrementa en una unidad.
- La estructura de control, *while* la cual está definida de la siguiente manera:

while *nombre* $\neq 0$ do; .
.
.
end;

Con estas tres instrucciones básicas, se pueden construir otras que hagan más legible el programa:

- La función cero, se implementa limpiando una variable.
  clear nombre;

  while *nombre* $\neq 0$ do;
  `decr` *nombre*;
  end;

- La asignación del valor de la **variable1** a la **variable2**, será representada por el siguiente código:

  clear aux;
  clear nombre1;
  while nombre2 $\neq 0$ do;
  `incr aux;`
  `decr nombre2;`
  end;
  while aux $\neq 0$ do;
  `incr nombre1;`
  `incr nombre2;`
  `decr aux;`
  end;

A partir del código anterior se implementan las *funciones recursivas parciales* que ya hemos visto, *combinación, composición, recursividad primitiva* y *minimalización*. De acuerdo a lo visto en la subsección anterior, las funciones anteriores son suficientes para calcular cualquier función recursiva parcial, por lo tanto, el lenguaje elemental descrito arriba proporciona el poder para platear una solución a cualquier algoritmo, sus limitantes serán la legibilidad y lo tardado de la implementación. Podriamos afirmar que lo que hemos codificado es una *máquina de Turing*, ya que ningún otro lenguaje podría encontrar la solución a algo más que las funciones recursivas parciales y si lo hiciera contradiría la tesis de Church-Turing.

En esta sección se buscó esclarecer el conjunto de problemas computables. Se vieron diferentes propuestas equivalentes debido a las conclusiones obtenidas en el estudio de los límites de la computación, resaltando la importancia de la máquina de Turing por definir el conjunto de procesos computables.

4 Problemas intratables

En primera instancia podríamos clasificar a los problemas en:

- *Computables* (decidibles)
- *No computables* (no decidibles)

Los problemas decidibles [1] se clasifican según los recursos (espacio y/o tiempo) que consumen para ser resueltos usando siempre su mejor solución algorítmica conocida para la peor entrada posible. Indecidibles, por su parte, aplica a una fórmula que no pertenece a un determinado sistema formal y que su negación tampoco pertenece al mismo. Esta noción está relacionada con la incompletitud del sistema.

Un ejemplo clásico de un problema indecidible es el ya menciaonado problema de Hilbert [5]:

> *Determinación de la solubilidad de una ecuación diofántica*: dada una ecuación diofántica con un número arbitrario de incógnitas y con coeficientes enteros racionales: idear un proceso por el que se pueda determinar mediante un número finito de operaciones si la ecuación es soluble en enteros racionales.

Este es el tipo de problemas que se plantearon en el programa formalista de Hilbert, dicho en un lenguaje menos técnico: establecer para las teorías matemáticas procedimientos de tipo finito para decidir si una fórmula dada es o no un teorema de la teoría [2], éste es el problema de decisión de la teoría o *Entscheidungsproblem*. Plantear este problema constituía el sueño de Leibniz de tener un *calculus ratiocinator* que decidiera sobre las verdades lógicas, mediante la reducción del razonamiento, al cálculo aritmético.

4.1 El problema indecidible de los mosaicos

El siguiente problema se basa en una tesela o baldosa, las cuales son estructuras geométricas cerradas como las de la figura 2.

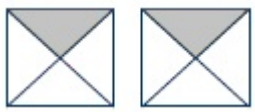

Figura 2: Imagen de un ejemplo de tesela Wang y su configuración de color

La entrada al problema es un número finito de teselas T. Cada tesela de T tiene su descripción definida por sus cuatro colores en un orden. El problema se plantea como a continuación:

Dada cualquier superficie finita, de cualquier tamaño, ¿puede recubrirse usando teselas de los tipos de T, de forma que las teselas adyacentes tengan el mismo color en los lados que se tocan? Además, existen las restricciones de que las teselas no se pueden rotar ni reflejar.

Nótese que hay un número ilimitado de baldosas de cada tipo, pero el número de tipos es finito.

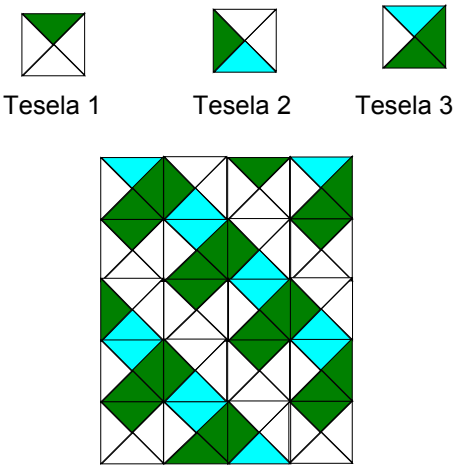

Figura 3: Representación del problema de dominó de Wang en una superficie.

En 1961, H. Wang [6] se interesó en encontrar un algoritmo capaz de determinar si era posible colocar las teselas, con las restricciones ya mencionadas, de forma que se llenara la superficie deseada. Wang en [7] afirmaba haber encontrado el algoritmo pero con la suposición de que los conjuntos de teselas que construyen un mosaico aperiódico también forman un mosaico periódico. Sin embargo, en 1966 Berger [8], alumno de Wang, descubrió el primer conjunto de teselas que sólo formaba mosaicos aperiódico, el conjunto constaba 20,156 teselas; el conjunto se redujo en 1971, cuando R. M. Robinson en [9] publicó el conjunto de 6 teselas con modificaciones en las esquinas que tampoco producía mosaicos periódicos pero si aperiódicos. Es importante mencionar que las teselas de Robinson son equivalentes a 36 teselas cuadradas que solo tienen movimiento de traslación [1]. Estos conjuntos determinaron que el algoritmo de Wang no era universal para todos los conjuntos posibles de teselas, por lo tanto el problema, de nuevo, no tenía un método que lo resolviera.

El anterior problema es un problema de decisión donde la respuesta a la que debería llegar un programa sería *si* o *no*, es un caso particular tipo dominó. Se le llama problema *indecidible* por ser imposible diseñar un algoritmo que dada una entrada T resuleva el problema.

[1] En 1977, R. Penrose cubrió un plano de forma aperiódica con solo dos poligonos, la flecha y el cometa [10].

4.2 Problema de la parada y verificación

Con el ejemplo anterior podemos introducir otros dos problemas muy conocidos por su fuerte grado de indecibilidad:

> **Problema de la parada**: dado un programa (o algoritmo) A y un valor de entrada X ¿se puede contruir un algoritmo para saber *siempre* si A parará o no?

Hasta el momento no existe un algoritmo para determinar si un programa, con cierta una entrada, parará o no. Tampoco existe un algoritmo para la solución universal del siguiente problema:

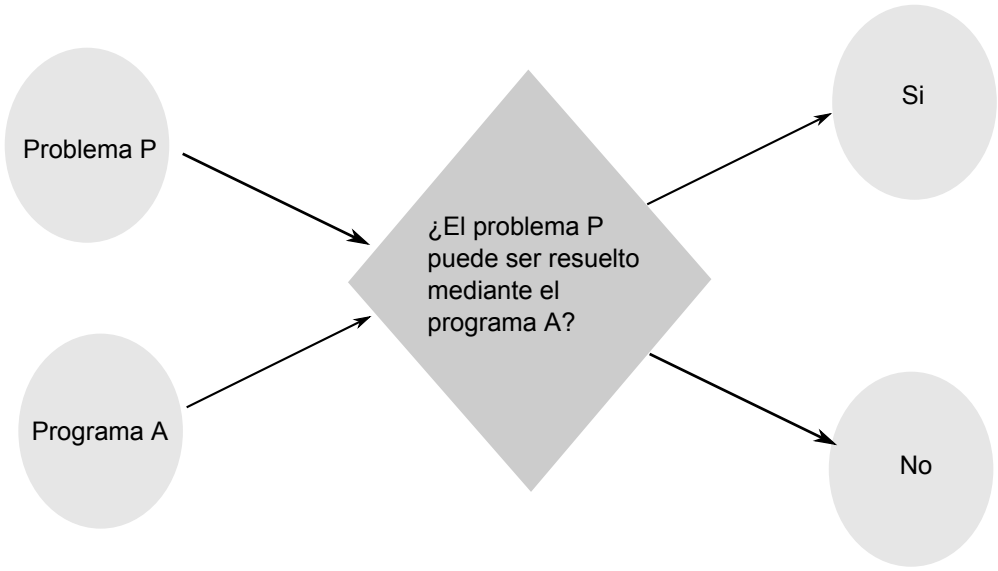

Figura 4: Problema de verificación

> **Problema de la verificación**: dada la descripción formal de un *problema* P y un programa (o algoritmo) A que supuestamente lo resuelve, ¿es posible contruir un algoritmo que verifique si A efectivamente resuelve P?

Este problema también es considerado *no computable* y su solución es más complicada que el problema de la paradada o el de los mosaicos por ser un problema más general, que no sólo se ocupa de saber si el algoritmo se detendrá sino además saber si lo resolverá.

4.3 Problemas decidibles

En el conjunto de los problemas decidibles se trabaja en reducir la cantidad de recursos que requieren para su solución. Los recursos que se usan comunmente

son:

- **Tiempo**: el cual se puede medir a *priori* o *posteriori*. El tiempo a priori se obtiene por medio de el número de acciones elementales mientras que con la técnica de posteriori el recurso se mide en tiempo (segundos, minutos, etc.). Este recurso es el prioritario en disminuir.
- **Ejecución**: es una medida espacial en la que se mide la cantidad de memoria necesaria para llegar a la solución.

La medida de la complejidad temporal depende de la entrada, comunmente denominada N. Ésta, de forma regular, influye de las siguientes maneras:

- Lineal: kN
- Polinomial: kN^m
- Logarítmica: $\log N$
- Exponencial: k^N

Donde la constante $k \in \mathbb{N}^+$. Por ejemplo, el problema de búsquedas sobre listas ordenadas tiene un grado de complejidad de $\log N$, esta es la *cota mínima* de complejidad para este *problema* debido a que esta demostrado que ningún algoritmo nuevo mejorará este grado. Por ello se considera un *problema cerrado*.

Aquellos problemas cuya solución es un algoritmo de complejidad polinomial (kN^m) son conocidos como problemas *tratables*. Por su contraparte, aquellos que su complejidad resulta ser super-polinómica, como la exponencial, se les llaman *intratables*.

Algunos ejemplos de problemas intratables son:

- Las torres de Hanoi 2^N.
- Juego de ajedrez 3^N.
- Aritmética de Presburguer $2^{2^{kN}}$ [11].

El ejemplo clásico de un problema intratable es el problema de *las torres de Hanoi*. El problema se centra en un sistema como el de la figura 5 con tres torres y un número N de discos, donde $N \in \mathbb{N}^+$. La trama consiste en mover los N anillos de la torre A a la torre B o C, usando una de las otras como ayuda pero sin que un disco de mayor tamaño quede encima de uno de menor tamaño.

Su cota mínima de complejidad es de 2^N, una computadora con la capacidad de 10^6 operaciones por segundo es capaz de resolver en 1 ms el problema con 10 anillos y casi 36 años con 50 anillos.

Existen otros problemas que, a la fecha, presentan una cota superior, por ejemplo es el *problema del viajero* o *TSP*, por sus siglas en ingles *(Traveling Salesman Problem)* [12], este tiene una complejidad $\frac{(N-1)!}{2}$ [13], donde $N \in \mathbb{N}^+$,

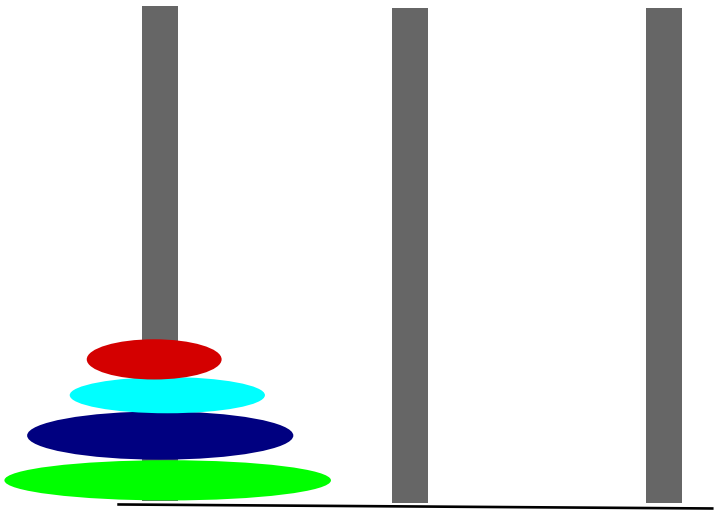

Figura 5: Torres de Hanoi con $N = 4$

por lo que tiene una explosión combinatoria que crece muy fácilmente, por ejemplo volviendo al procesador con capacidad de 10^6 operaciones por segundo este es capaz de resolver en 3.62 segundos el problema con 10 ciudades y casi en 1,928 años con 20 ciudades. Estos problemas se conocen como *NP-Completos*, estos tienen la propiedad de que su cota acutal se considera no tratable con una máquina *determinista*, además, aun no se comprueba que no se pueda llegar a la solución con una *determinista*, pero con una máquina *no determinista* es posible obtener un resultado óptimo.

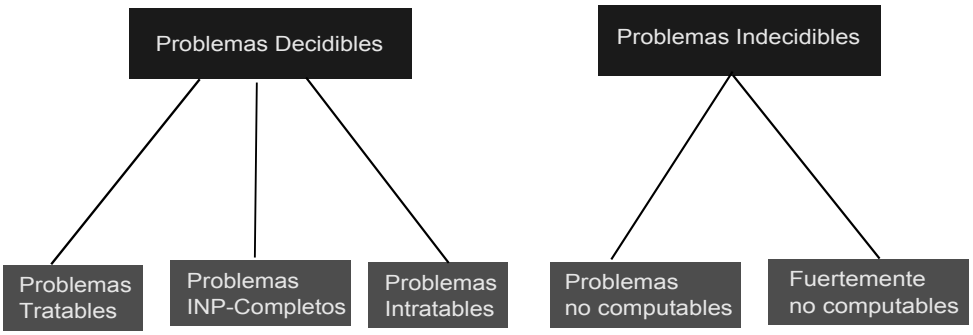

Figura 6: División general de los problemas

5 Teoría de autómatas

La teoría de autómatas se dedica al estudio y desarrollo de modelos abstractos que representan formalmente una computación o un algorítmo, cada autómata es capaz de resolver un conjunto de problemas. Los orígenes de esta área de estudio se desarrollaron en áreas que no parecían tener una relación tan directa con la ciencia de la computación, de hecho, sus contribuciones iniciales datan de fechas anteriores a la máquina de Turing, la teoría de incompletitud y la contrucción de la primera compuerta AND. En 1921, Emil Post publicó en su tesis doctoral [15] la introducción a un *marco* como sistema de inferencia basado en un proceso finito de manipulación de símbolos para sistemas lógicos [14]. Post utilizó el principio puramente sintáctico de *modus ponens* en *Principia Mathematica* para derivar una fórmula del lenguaje con las reglas del mismo. Con lo anterior Post encontró lo que hoy llamamos *cálculo proposicional* y las operaciones formales que denominó *producciones*. Las producciones de Post fueron aplicadas por Noam Chomsky para producir la teoría de las gramáticas de los lenguajes naturales, lo que derivó en la jerarquía de lenguajes de Chomsky. La relación que hay entre Post, la teoría de gramáticas y la ciencia de la computación se manifesto cuando John Backus, independiente del trabajo de Chomsky, utilizó las producciones de Post para crear una sintaxis para el desarrollo de *ALGOL 58*, fue entonces cuando se evidenció que los lenguajes que se podían escribir en términos de la sintaxis de Backus eran los mismos que los lenguajes independientes del contexto de la jerarquía de Chomsky. Después, con el desarrollo de los autómatas surgió otra relación cuando un *autómata finito* reconocía la misma clase de lenguaje que una de las jerarquías de Chomsky, los *lenguajes regulares* [16].

Para el estudio de esta campo, se da una introducción a los siguientes conceptos para su mejor entendimiento:

- *Símbolo*: es una representación de un objeto, puede ser cualquier caracter que represente un elemento, como: 5, β, n, etc.

- *Alfabeto*: es un conjunto finito de símbolos. Éste conjunto no puede ser vacío. El alfabeto comunmente se representa por Σ.

- *Cadena*: es una secuencia de símbolos contruida a partir de los elementos del *alfabeto*, también se le llama *frase* o *palabra*. Se representa por una letra griega como: α, β, ω. Las cadenas conforman el conjunto infinito del *alfabeto universal* denotado por Σ^* el cual contiene todas las combinaciones finitas de los símbolos del alfabeto. Por ejemplo, del alfabeto $\Sigma = \{1, 0\}$ se obtiene la cadena $\omega = 100$ que forma parte del conjunto $\Sigma^* = \{1, 0, 11, 00, 01, 10, 000, 100, ...\}$. La cadena vacía se representa por ϵ.

- *Lenguaje*: es un conjunto finito de *cadenas* formadas por los símbolos del *alfabeto* que cumple ciertas restricciones propias de cada lenguaje. Por ejemplo el lenguaje $L = \{100, 001, 00, 111\}$ se forma con $\Sigma = \{1, 0\}$ pero la

cadena $\omega = 1010$ no forma parte del lenguaje L, la notación para designar lo anterior se escribe $\omega \notin L$.

5.1 Autómatas finitos

En la teoría de autómatas, el modelo más simple de un autómata es el *autómata finito* [17]. Éste es un mecanismo matématico que acepta infromación de *entrada*, *la procesa* por medio de cálculos o transformaciones simbólicas generando una *salida*. Este modelo puede representar una función cuyos argumentos son la entrada y el resultado la salida. Su funcionamiento consiste en ir pasando de un estado a otro, en función de los caracteres de la cadena de entrada que recibe, hasta llegar a un estado final, el cual indica que la máquina ha llegado a un estado de *aceptación* o de fin. Este autómata puede ser representado por un diagrama de estados como el de la figura 7

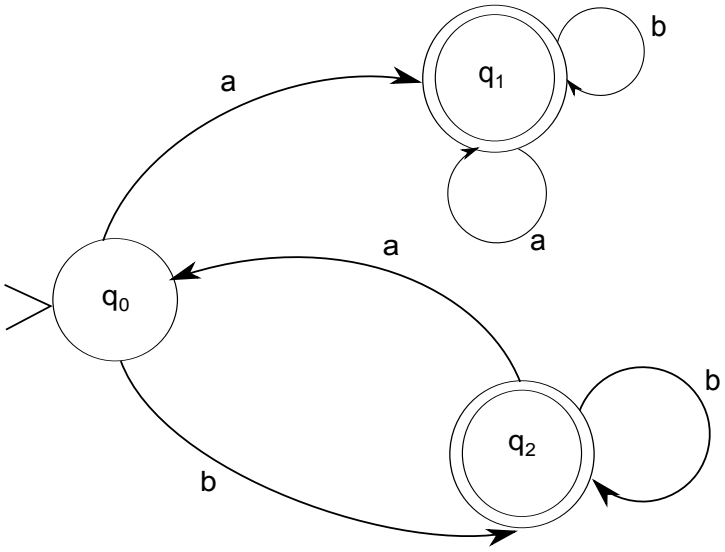

Figura 7: Diagrama de estados de un AFD

El diagrama de la figura 7 nos sirve para ejemplificar el procesamiento que se lleva acabo con la cadena de entrada *bbaa*. Inicia el autómata en su estado inicial q_0, cuando recibe el primer caracter b pasa al estado q_2 de acuerdo a la transición que se marca en el diagrama con una flecha de q_0 a q_2 cuando una b es recibida en q_0. Con esta misma lectura del diagrama vemos que al recibir la siguiente b en q_2 este permanece en el mismo estado q_2, al llegar el siguiente caracter a se regresa al estado q_0 y al ingresar el siguiente caracter a se desplaza al estado q_3 el cuál es un estado final o de aceptación. Con lo anterior podemos inferir que la cadena de entrada es una cadena válida del lenguaje propio de este autómata. Otro ejemplo para una cadena aceptada por este lenguaje es *baaabb*, la siguiente secuencia muestra su validación: $q_0 \Rightarrow_b q_2 \Rightarrow_a q_0 \Rightarrow_a q_1 \Rightarrow_a q_1 \Rightarrow_b q_1 \Rightarrow_b q_1$, la cadena ha

sido completamente leida y el autómata quedó en un estado de aceptación, esto indica que la cadena es válida para el lenguaje de dicho AFD. En un caso contrario al anterior, la cadena *bbba* no forma parte del lenguaje de dicho autómata debido a que, siguiendo la secuencia $q_0 \Rightarrow_b q_2 \Rightarrow_b q_2 \Rightarrow_b q_2 \Rightarrow_a q_0$ que representa el procesamiento de la cadena, el autómata termina su proceso en un estado que no es de aceptación y por lo tanto la cadena no forma parte del lenguaje. Es necesario resaltar que los *estados* son el único medio que le proporciona memoria al autómata, por ejemplo, si nos encontramos en el estado q_2 sabemos que al menos hemos recibido un caracter b.

Los autómatas tienen una representación formal que se define a continuación:

> *Definición*: una autómata de estados finitos M es una quíntupla $(\Sigma, Q, s, F, \delta)$, donde:

- Σ es el alfabeto, el conjunto finito de símbolos.
- Q es la colección finita de estados.
- s es el estado inicial, s $\in$ Q.
- F es el conjunto finito de estados finales, pueden ser uno o más, F $\subseteq$ Q
- δ representa la función de transición en la que a partir de un estado y un símbolo de entrada se obtiene un nuevo estado $Q \times \Sigma \rightarrow Q$.

Existen dos clases de autómatas finitos, los *deterministas (AFD)* y los *no deterministas (AFN)*. La primera de estas clases tiene la característica de que estando en un estado q y con un símbolo de entrada determinado, el siguiente estado siempre será el mismo. Esto se logra con la restricción de que cada estado sólo puede tener una transición a otro estado con cada símbolo de Σ. Los autómatas finitos no deterministas no siguen la anterior restricción, a éstos se les permite que cada estado tenga cero o más transiciones con cada símbolo del alfabeto, además, la transición puede ser provocada no sólo por un símbolo, sino también por subcadenas o hasta ϵ. Por todo lo anterior, los AFN no tienen una función de transición sino una relación. El autómata se define formalmente como una quíntupla $(\Sigma, Q, s, F, \Delta)$, donde Σ, Q, s, F representan lo mismo que el AFD y Δ es la relación de transición $Q \times \Sigma^* \times Q$, esta notación representa una tripleta (q_1, ω, q_2), en la que el segundo elemento de la relación es una palabra que se consume permitiendo ir de q_1 a q_2. Los AFD son un subconjunto de los autómatas no deterministas, por lo tanto, todo AFD es un AFN.

La definicón formal del autómata de la figura 7 es $M = (\Sigma, Q, q_0, F, \delta)$, donde:

- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1, q_2\}$
- $F = \{q_1, q_2\}$

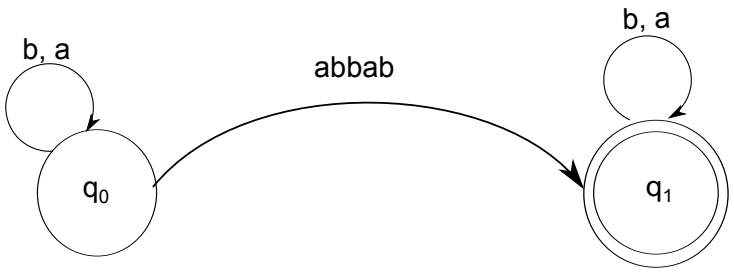

Figura 8: Diagrama de estados de un autómata no determinista

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

Para ejemplificar un AFN se tomará el de la figura 8 el cual se define formalmente como $M = (\Sigma, Q, q_0, F, \Delta)$, donde:

- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1\}$
- $F = \{q_1\}$
- $\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, abbab, q_1), (q_1, a, q_1), (q_1, b, q_1)\}$

Ambas clases de autómatas aceptan una cadena cuando han consumido toda la cadena de entrada, siguiendo las transiciones que lo definen, y, además, al último estado que llegaron es un estado final de F.

El lenguaje aceptado por un autómata M es el conjunto de las palabras aceptadas por dicho autómata. En el caso del ejemplo del AFD de la figura 7 el lenguaje de dicho autómata son todas las palabras que empiezan con a, que contienen aa o que terminan con b como $aaab$, baa, aba. Este lenguaje no acepta la cadena vacía ϵ, para que esto ocurra se necesita que el estado inicial sea de aceptación. En el ejemplo del AFN de la figura 8 el lenguaje de este autómata es el conjunto de todas las cadenas con cero o más a's o b's antes o después de la subcadena $abbab$, en una notación formal el lenguaje del AFN en cuestión se define como $L = \left\{\{a, b\}^* \cup abbab \cup \{a, b\}^*\right\}$.

Los autómatas finitos que hemos definido realizan la tarea de aceptar o rechazar una cadena, pero tambien existen AF con la propiedad de producir una salida diferente a "si" o "no". Hay dos formas de definir la salida de un AF, en la primera forma la salida depende de las transiciones y en la segunda depende del estado en el que se encuentra el AF. Estos AF corresponden al autómata de Mealy y al autómata de Mooree respectivamente. Propuestos por G. Mealy y E. Moore [17].

Las autómatas de Moore tienen la característica de que su salida depende del estado en que se encuentra. La salida es producida cada vez que el autómata llega a un estado ya que cada estado puede tener asociado una salida.

En los autómatas de Mealy, por su parte, la salida se produce en la transición de un estado a otro estado o en un bucle. A diferencia de Moore, en este autómata cuando uno llega a cierto estado la salida puede no ser siempre la misma debido a que la salida también depende del estado anterior. Cada transición marcada en el diagrama de estados puede producir una salida diferente.

En la jerarquía de lenguajes de Chomsky el lenguaje que es aceptado por los autómatas finitos es el lenguaje regular, el conjunto de lenguajes más pequeño de la jerarquía. El métdo que valida que un lenguaje regular es aceptado por un AF, y que un AF representa un lenguaje regular, consiste en una serie de transformaciones que son aplicadas a una expresión regular para convertirla en un autómata finito que acepte el mismo lenguaje que la expresión. Dichas transformaciones se pueden representar con las *gráficas de transición*. En caso de que sea necesario concer con más a detalle el método de validación, se recomienda consultar la siguiente referencia [17].

5.2 Autómatas de pila

En la jeraquía de Chomsky los lenguajes regulares son los más básicos y están contenidos en su totalidad por el siguiente nivel, el conjunto de los lenguajes libres de contexto, estos serán referidos por las siglas *LLC*. En este nivel de la jerarquía los AFD no tienen la capacidad de aceptar todos los LLC. Una propiedad que debería tener un autómata que acepte todos los LLC es recordar cadenas arbitrarias de caracteres, para logralo se implementa un almacenamiento auxiliar denominado *pila*, donde se pueden depositar, caracter por caracteres, cadenas arbitrarias. A este modelo se le llama *autómata de pila*, al que nos referiremos por *AP*.

La pila es una estructura en la que se almacenan caracteres uno sobre otro evitando que el primero que se introdujo salga antes que todos los que fueron introducidos después y asegurando que el último que entró en la pila sea el primero en salir. Al iniciar el procesamiento de una cadena la pila se encuentra vacía y en el transcurso de la operación se van almacenando o sacando caracteres de la pila, según lo marquen las transiciones, hasta elfinal del procesamiento.

Para que la cadena sea aceptada, el AP debe cumplir los siguientes criterios:

- La palabra de entrada se debe haber leido completamente
- El AP debe encontrarse en un estado final al final del precesamiento
- La pila debe estar vacía

Un ejemplo de un AP es el que se muestra en la figura 9. Su representación es muy parecida al diagrama de estados de un AF pero con una modificación en las transiciones. La transición sigue esta notación $\omega/\alpha/\beta$, donde ω es el caracter leido de la cadena de entrada, α es el elemento o la secuencia de caracteres que se saca de la pila (acción típicamente conocida como *Pop*) y β es el elemento o la secuencia de caracteres que se introduce en la pila (acción típicamente conocida como *Push*). Para representar que no se consume o se introduce ninguna cadena

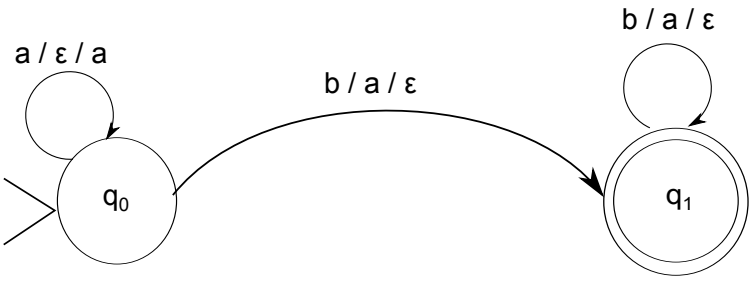

Figura 9: Representación de un autómata de pila

a la pila se uiliza en la notación el ϵ. Teniendo estas características en cuenta, podemos ver que lenguaje es el que acepta nuestro AP de la figura 9. La transición $a/\epsilon/a$ se produce cuando el caracter leido es una a, no extrae nada de la pila e inserta una a en la pila. La siguiente transición $b/a/\epsilon$ se produce cuando es leida una b entonces saca una a del tope de la pila y no mete nada en ella. Anilizando todas las transiciones y su colocación se llegaría a la conclución de que el AP acepta cadenas de la forma $a^n b^n$, donde $n \in \mathbb{N}$. Tomemos como ejemplo la cadena $aabb$ para ver si es aceptada por el autómata, en la tabla de la figura 10 se representa el procesamiento.

Estado	Caracteres por leer	Pila	Transición
q_0	aabb	ε	
q_0	abb	a	1
q_0	bb	aa	2
q_1	b	a	3
q_1	ε	ε	4

Figura 10: Procesamiento de la palabra $aabb$ por un AP

El procesamiento que se muestra en la figura 10 finaliza en un estado de aceptación del autómata, la pila queda vacía y la cadena de entrada se leyó en su totalidad, con estos tres aspectos cumplidos concluimos en que la palabra forma parte del lenguaje del AP de la figura 9.

La representación formal de un AP es la siguiente:

Definición: una autómata de pila M es una séxtupla $(\Sigma, Q, \Gamma, s, F, \Delta)$,

donde:

- Σ es el alfabeto de entrada, el conjunto finito de símbolos.
- Q es la colección finita de estados.
- Γ es el alfabeto de la pila.
- s es el estado inicial, s $\in$ Q.
- F es el conjunto finito de estados finales, pueden ser uno o más, F $\subseteq$ Q
- Δ representa la relación de transición en la que se indica los elementos a sacar y meter en la pila y el un nuevo estado del autómata $\Delta \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*)$.

La notación de las transiciones de Δ es $((p, u, \beta), (q, \gamma))$, esto implica que estando el estado p y habiendo consumido la subcadena u se extrae la subcadena β de la pila, se posiciona en el estado q y se introduce la subcadena γ en la pila.

Con lo anterior formalizaremos el ejemplo del AP de la figura 9 el cual es un séxtuplo $M = (\Sigma, Q, \Gamma, q_0, F, \Delta)$, donde:

- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1\}$
- $\Gamma = \{a, b\}$
- $F = \{q_1\}$
- $\Delta = \{((q_0, a, \epsilon), (q_0, a)), ((q_0, b, a), (q_1, \epsilon)), ((q_1, b, a), (q_1, \epsilon))\}$

Un AP tiene la capacidad para aceptar los lenguajes regulares y los LLC de la jerarquía de Chomsky. Acepta los regulares debido a que un AF es un AP pero que no hace uso de la pila, por lo que sus transiciones no sacan ni intruducen nada en la pila y en la referencia [17] se encuentra la prueba de que si un lenguaje L es LLC, entonces existe un AP M tal que $L(M) = L$.

5.3 Máquinas de Turing

En la sección 3 se habló de la máquina de Turing y su capacidad de procesar el conjunto de problemas computables, sin embargo, no se definieron sus caracteristicas ni su descripción formal. En esta subsección se verán las propiedades y mecanismos de una *maquina de Turing*.

La máquina de turing fue propuesta en los años treinta como una solución para trabajar problemas que podían ser estructurados en pasos bien definidos y finitos. La MT (como se le abrevia comunmente) es un mecanismo abstracto que consta de una cabeza lectora-escritora y una cinta con caracteres de entrada, la cinta tiene una longitud infinita hacia la derecha pero hacia la izquierda no es infinita, debido a que contiene un cuadro en el extremo izquierdo que contiene el símbolo $\sqcup$ que representa el caracter blanco. La cabeza se puede mover en

ambas direcciones (izquierda y derecha) pudiendo paras varias veces sobre el mismo elemento de la cinta. Al iniciar la operación la MT, la cabeza lectora está colocada en el primer elemento de la cinta, el espacio blanco. La secuencia de operación que sigue la MT al procesar una entrada es la siguiente:

1. Leer un elemento de la cinta
2. Cambiar de estado
3. Realizar una acción sobre la cinta

Las acciones que ejecuta la MT en el paso 3 sólo se pueden aplicar una a la vez, estas son:

- Escribir un símbolo en la cinta.
- Mover la cabeza en alguna dirección

El primer elemento de la cinta siempre es el caracter blanco, en la segunda posición empieza la cadena de entrada y del último caracter de la cadena de entrada hacia la derecha se llena el resto de los espacios de la cinta con el caracter blanco $\sqcup$. La MT termina su operación cuando se alcanza el estado llamado *halt* representado por h, se puede decir que *halt* es el único estado de aceptación con la limitante de que h no tiene ninguna transición, cuando se llega es porque a finalizado completamente el procesamiento e implica que la cadena ha sido aceptada. Si la cadena que se introduce no es propia del lenguaje del MT, este se quedará atrapado en un ciclo infinito. Al igual que en los autómatas anteriores, en este también el lenguaje que acepta es el conjunto de palabras aceptadas por dicho MT.

Para graficar el funcionamiento de la MT se tomará la siguiente notación para indicar las acciones que realizará la máquina, cuando en la flecha de transicón se indique σ/L, significa que la cadena de entrada es σ y que la cabeza lectora hace un movimiento a la izquierda, cuando en vez de L tenga una R significará que el movimiento que realiza la cabeza lectora es un desplazamiento hacia la derecha, por sus siglas en inglés $Left$ y $Right$; cuando lo que se requiere es escribir en la cinta, la acción se representará por σ/ξ, donde ξ es un caracter.

Con los elementos anteriores podemos diseñar una MT que acepte palabras con la forma $\{(a,b)^* \cup a\}$, esto se refiere a cadenas de *a's* y/o *b's* con una a como caracter final. Para su verificación, cuando la cadena se haya leido completamente y la cabeza lectora se encuentre en el espacio blanco a la derecha de la cadena, se regresará a la última letra, con una acción L, y verificará si el último caracter es una a. La representación gráfica se muestra en la figura 11.

La MT tiene una representación formal, en la que una MT es un quíntuplo $(\Sigma, Q, \Gamma, s, \delta)$, donde:

- Σ es el alfabeto de entrada, el conjunto finito de símbolos, donde $\sqcup \notin Q$.
- Q es la colección finita de estados, en la que $h \in Q$.

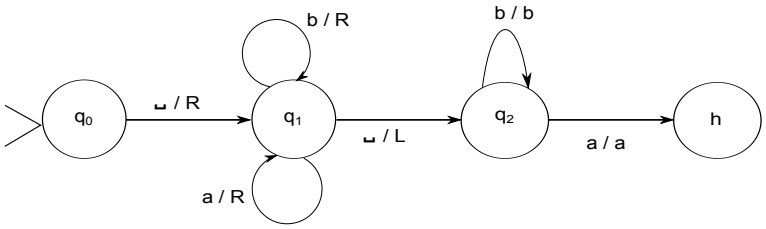

Figura 11: Representación de una MT que procesa cadenas compuestas de a's y b's pero que finalizan con una *a*.

- Γ es el alfabeto de la pila, donde $\sqcup \in \Gamma$ y $\Sigma \subseteq \Gamma$.
- s es el estado inicial, s $\in$ Q.
- δ representa la función de transición en la que se indica el un nuevo estado de la MT y su acción a realizar $(Q - \{h\} \times \Gamma) \rightarrow K \times (\Gamma \cup \{L, R\})$.

En δ se denota que dependiendo de un estado, excepto h, y el caracter leído en la cinta se obtiene un estado nuevo y una acción a realizar, sea escribir o mover la cabeza en una dirección. De esta forma, cada transición puede ser escrita como $\delta\,(q, a) = (p, b)$, donde q es el estado actual; a es el caracter sobre el que se encuentra la cabeza lectora; p es el nuevo estado y b es el caracter que se escribirá. También puede escribirse en lugar de b, L o R denotando un movimiento hacia la izquierda o derecha.

Decimos que un lenguaje L es *Turing-aceptable* si hay alguna MT que llega al estado *halt* para toda entrada $\omega \in L$.

Además de procesar palabras, una MT también puede procesar funciones, y un caso particular de funciones es aquel en que el resultado está limitado a *sí* o *no*, tal resultado denota si una función tiene cierta propiedad o no. Un caso de este tipo de funciones es *el problema de paro* en el que dada una cadena ω y la descrición de un lenguaje L, decidir si $\omega \in L$, este problema también es conocido como el de la *correspondencía de la palabra* estudiado por Post. El estudio de este problema ha determinado que es un problema que no puede ser resuelto con una MT y por lo tanto por ninguna otra máquina [17].

Los límites de la máquina de Turing tiene su fundamento en el estudio de los límites de los lenguajes que la MT tiene la capacidad de procesar. Dentro de la jerarquía de Chomsky, el lenguaje que contiene a los lenguajes antes mencionados es el lenguaje recursivamente enumerables, también llamado lenguaje estruturado por frases, y es asociado con la MT debido a que es la única máquina con las propiedades necesarias para procesar ese lenguaje. Es importante recalcar que existen otros lenguajes, pero el estudio de los lenguajes relacionados con las máquinas se limita a los recursivos enumerables porque es el más extenso conjunto de lenguajes que pueden ser analizados por medio de algorítmos [17]. La razón por la cuál se afirma que hay lenguajes que no se pueden analizar con una MT es porque el número de máquinas de Turing basadas en un alfabeto Σ

es *infinito contable* porque existe una biyección con el conjunto de los números naturales, en cambio, el número de cadenas, y por lo tanto de lenguajes, que se obtiene a apartir del conjunto Σ es *infinito no numerable*, ello conlleva que que el conjunto de MT es menor que el conjunto de los lenguajes basados en un alfabeto Σ. Por lo tanto existen lenguajes basados en Σ que no son estructurados por frases.

Podemos ver una una reflexión importante de las capacidades de la MT con respecto a la inteligencia artificial ya que existe la creencia de que el cerebro humano trabaja ejecutando algoritmos, lo que nos llevaría a la conclusión de que el cerebro humano no comprende ciertos lenguajes al igual que las MT, pero existe otra tendencia que piensa que el nivel de capacidad del cerebro es mucho más poderoso que la ejecución de algirtmos, por lo tanto, con esta teoría, la computadora jamás podría tener una verdadera inteligencía.

Existen otros modelos diferentes a las MT e incluso hay MT con diferentes características que una MT tradicional, entre ellas podemos mencionar los siguientes ejemplos:

- MT con varias cabezas y/o cintas.
- MT no determinista.
- Máquinas de Post.
- Gramáticas.

Todos estos modelos o implementaciones, al igual que una MT, no pueden realizar un anális sintáctico a aquellos lenguajes que están en un nivel superior a los lenguajes estructurados por frases.

Para finalizar la sección se comentará un concepto de gran relevancía para las ciencias computacionales que Turing sostuvo en su tesis, el concepto de la *máquina universal de Turing*. Esta máquina está diseñada para ejecutar programas que se almacenan en sus cintas, dichos programas son una versión *codificada* de una MT que realiza la tarea que desea que se ejecute en la máquina universal. La codificacón de la MT que realiza el procesamiento es realizada en el lenguaje universal compuesto de cadenas de 0,1, por lo tanto, para que una MU (máquina universal) ejecute un proceso lo primero que es necesario hacer es diseñar la MT que hace el proceso, después se codifica en el lenguaje universal y por último intrucimos en la cinta la codificación de la MT. La MU ejecutará las transiciones y manipulará los datos en la cinta. En algunas lecturas se puede encontrar una representación de la MU con 3 cintas que simplifica su comprensión. Una de las cintas es usada para almacenar el programa de entrada y los datos; una de las restantes se emplea para manipular los datos y en la última se almacena una representación del estado actual de la MT que se ejecuta. Se recomienda acudir a la referencía [1] para más detalles de la máquina universal.

6 Conclusiones

Historicamente es imprescindible el desarrollo no sólo matemático sino totalmente interdiciplinario para comprender problemas cada vez más complejos, ya que intuimos que la relación con las demás áreas es inherente cada vez que atacamos un problema con mayor profundidad. Hemos visto que en diferentes diciplinas se han estudiado las capacidades de los sistemas formales para describir la solución de todos los problemas y que los trabajos más importantes en el área han llegado a conclusiones equivalentes que actualmente nos siguen marcando el mismo límite para utilizar todo el avance tecnológico que se ha desarrollado. Mientras la tesis de Church-Turing siga sin ser refutada, las capacidades de la computación serán las mismas independientemente de los recursos o modelos que se utilicen, estos pueden, en el mejor de los casos, mejorar en tiempo y recursos los problemas intratables pero no cambiarán los problemas no decidibles por decidibles, sin embargo, debe coexistir ese trabajo interdiciplinario que nos lleva por nuevos caminos como el de la computación natural que nos provee de nuevos paradigmas para tratar problemas más grandes o complejos. Por su parte, el concepto de universalidad que Turing desarrolló es lo que hoy tenemos físicamente como una computadora, esa máquina que ejecuta programas codificados en el lenguaje universal.

7 Agradecimientos

Quiero expresar un profundo agradecimiento a la academia mexicana de ciencias, AMC, por el interés, empeño y esfuerzo que demuestra al permitir a estudiantes, que tenemos el deseo de conocer mucho más para mejorar académica y personalmente, la oportunidad de trabajar y desarrollarnos con investigadores de trayectoría y trabajo sobresaliente. Además del enlace con investigadores que nos motivan a seguir trabajando con esfuerzo y dedicación en la ciencia, y con instituciones que nos abren las puertas y nos ofrecen los recursos necesarios para un trabajo de calidad, tambíen le doy las gracias por el apoyo económico que nos otroga para realizar una estancia productiva y amena.

References

[1] Glenn, B. J. (1993). *Teoría de la Computación. Lenguajes formales autómatas y complejidad.* Addinson-Wesley iberoamericana.

[2] Hofstadter, R. D. (1987) *Gödel, Escher, Bach: un Eterno y Grácil Bucle.* Tusquets Editores.

[3] Gutiérrez, O. J. (2008). *Máquinas de Estados Finitos. Breve Introducción.* Escuela Superior de Cómputo I.P.N.

[4] Feynman, R. C., & Feynman M. (2003). *Feynman Lectures on Computation.* Crítica editorial.

[5] López de Lacalle, J. (2002). "X problema de Hilbert y otros problemas no computables." *IV seminario de matemática discreta*, Universidad politécnica de Madrid.

[6] Alcalde, F. C., & González, P. S. (2008). "Espacios foliados definidos por mosaicos". *Rev. Semin. Iberoam. Mat. 3*, 3-32.

[7] Wang, H. (1965). "Games, Logic, and Computers". *Scientific American Vol. 213*, 98-106.

[8] Berger, R. (1966). "The undecidability of the domino problem." *Mem. Amer. Math. Soc. no. 66.*

[9] Robinson, R. M. (1971). "Undecidability and nonperiodicity of tilings in the plane." *Inventiones Math., 12*, 177-209.

[10] Penrose, R. (1974). "The role of aesthetics in pure and applied mathematical research." *Bull. Inst. Math. Appl., 10*, 266271.

[11] Fischer, M. J., & Rabin, M. O. (1974). "Super-Exponentional complexity of Presburger arithmetic." *Mac Technical Memorandum, 43.*

[12] Hincapié, R. I., Ríos, C.P., & Gallego, R. R. (2004). "Técnicas heurísticas aplicadas al problema del cartero viajante (TSP)." *Scientia et Technica, 24.*

[13] Moreno, A., Armengol, E., Béjar, J., Belanche, L., Cortés, U., Gavaldá, . . . Sánchez, M. (1994). *Aprendizaje Automático*, Ediciones UPC - Universidad Politécnica de Catalua.

[14] O'Connor, J. J., & Robertson, E. F. http://www-groups.dcs.st-and.ac.uk/ history/Biographies/Post.html.

[15] Post, E. L. (1921). "Introduction to a General Theory of Elementary Propositions." *American Journal of Mathematics, 43*, 163-185.

[16] Davis, M. (1988). "Influences of mathematical logic on computer science." *A half-century survey on The Universal Turing Machine*, p.315-326.

[17] Brena, R. (2003). *Autómatas y lenguajes. Un enfoque de diseño*, Tecnológico de Monterrey. Mty.