METODOLOGÍA PARA EL DISEÑO Y USO EFICIENTE DE LOS RECURSOS ESPACIALES Y TEMPORALES EN LA SOLUCIÓN DE PROBLEMAS EN SISTEMAS ARIMÉTICOS DIGITALES

XILIANA PINEDA VANEGAS

cc. 1088300352

Proyecto de grado como requisito parcial para

obtener el título de Ingeniera Electrónica.

Director

MAURICIO HOLGUÍN L, Ing. M.Sc, Ph.D(C)

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

PEREIRA 2017

Metodología para el diseño y uso eficiente de los recursos espaciales y temporales en la solución de problemas en sistemas aritméticos digitales.

Xiliana Pineda Vanegas

cc.1088300352

Proyecto de grado como requisito parcial para

obtener el título de Ingeniera Electrónica

Director:

Mauricio Holguín L, Ing. M.Sc, Ph.D(C)

Universidad Tecnológica de Pereira

Programa de Ingeniería Electrónica

Pereira, 2017

Agradecimientos

A mis padres, por el apoyo y la paciencia que han tenido todo el tiempo, no solo universitario sino desde el colegio. A mis hermanos por el impulso, tiempo y energía transmitida. A mi director de proyecto de grado, Ing. Mauricio Holguín, por la paciencia y tiempo dedicado para el desarrollo de este proyecto. A “Don Gus”, por ser él y brindarme un espacio con el cual contar siempre sin importar qué o quién estuviera. A mis compañeros, no solo de la carrera, sino también a los que me crucé y pude compartir con ellos. A aquel profesor que en mi primer semestre me dijo que no iba a ser capaz con electrónica, que me vería y me iría mejor en otra carrera y a esos dos compañeros que me dijeron que mejor me cambiara de carrera porque se me dificultaba una materia; por ustedes me di cuenta que para ser Ingeniera no se necesita ser buena en todo, sólo tener una excelente actitud para hacer lo necesario.

Tabla de contenido

TABLA DE CONTENIDO DE FIGURAS

PARTE I.

INTRODUCCIÓN

CAPÍTULO 1. PLANTEAMIENTO DEL PROBLEMA

En el mundo los números binarios son de gran relevancia al ser usados con frecuencia, gracias a que dieron origen a la calculadora moderna y más delante al computador personal; esto se debe a la simplificación del lenguaje, que logra un nivel bajo de error, pues se basa en dos únicas respuestas excluyentes, como lo son el 1 y 0. El sistema binario ha sido la base de los sistemas digitales, dado que cuando se trabaja con un sistema digital, la respuesta e interpretación es obtenida de forma numérica.

En los sistemas digitales, un sistema complejo hace referencia a un conjunto de diferentes sub-sistemas, que al interrelacionarse entre ellos se obtiene un sistema de mayor nivel; la complejidad digital está compuesta por la complejidad circuital y la complejidad temporal, donde además la complejidad circuital, se relaciona con el uso de los recursos de espacio y memoria [1] y puede hacer referencia a la complejidad aritmética en las operaciones básicas, como lo es la adición y la multiplicación. Al hacer mención de las operaciones aritméticas y su complejidad en los métodos enseñados en la academia, se presenta uno de los problemas a investigar, ya que los métodos dados son los más conocidos y usados en general, pero eso no significa que sean los más eficientes, ya que normalmente se requiere que el sistema determine parámetros relacionados a la eficiencia y la simplicidad. La eficiencia se observa en el uso de los recursos de los cuales dispone dicho sistema, normalmente están los espaciales y los temporales; al hablar de la eficiencia espacial, se hace referencia al uso de memoria requerida para dar solución al problema y al mencionar del recurso temporal se refiere al tiempo obtenido para dar solución [2].

Otra de las necesidades parte de la implementación de sistemas de forma física, ya que cuando se hace de forma teórica o de simulación, los sistemas tienden a ser lineales, pero en su despliegue final se efectúan cambios en las diferentes variables del circuito; por lo tanto, si los métodos no son los más idóneos y se observa deficiencia en la simulación, a la hora de la implementación estos serían más notorios [3].

Por otra parte, en los sistemas digitales la lógica booleana es muy importante ya que define la forma de intersección entre los conjuntos; cuando un circuito está diseñado para una función booleana "$f \epsilon B$", normalmente se asume que es un sistema simplificado. Sin embargo, la mayoría de los diseños de circuitos conducen a éstos realizando una secuencia de funciones, por ejemplo, los sumadores típicos son secuencia de sumadores, por lo tanto, se observa una necesidad de investigar diferentes métodos en los cuales se pueda observar una eficiencia en los pasos u operaciones necesarios para obtener una respuesta [1].

Con base en lo anterior, este proyecto se trata de dar respuesta a sí es posible encontrar métodos que sean eficientes y óptimos para el uso de los diferentes tipos de recursos espaciales y temporales en la solución de un problema de sistemas digitales, teniendo en cuenta su complejidad circuital.

CAPÍTULO 2. JUSTIFICACIÓN

En la electrónica actual, los sistemas digitales desempeñan un papel muy importante en la vida cotidiana, ya que se está en un período del boom tecnológico; anteriormente las aplicaciones de la electrónica se limitaban a la informática, pero en la actualidad los sistemas digitales tienen una gran área de instrumentación, como son en las computadoras, transporte, entretenimiento, en la exploración del espacio, comunicaciones, instrumentación médica, control en procesos industriales, entre otras [3]. Un sistema digital es una composición de dispositivos diseñados para el uso de información lógica y o cantidades físicas, las cuales se representan en forma digital; eso quiere decir que los valores se obtienen de forma discreta. Por lo general se hace uso en los dispositivos electrónicos, pero también pueden ser mecánicos, neumáticos o magnéticos. Algunos de los sistemas digitales más comunes son las computadoras, las calculadoras digitales, los equipos de audio y video digital y el sistema telefónico; el cual es el sistema digital más grande del mundo [1]. Para llegar hasta este nivel de aplicación, se necesita entender y comprender lo más básico, en que se basa la electrónica digital, donde su base es el sistema binario. Cuando se habla de lo básico se puede hacer referencia a las operaciones aritméticas fundamentales y lo que hay detrás de ellas, como son el álgebra de Boole, los circuitos combinacionales y la implementación de las compuertas lógicas, donde se basa el presente documento. Al hacer referencia a la investigación de la eficiencia y optimización de la complejidad de operaciones aritméticas en los números binarios, en el cual se va a plantear el porqué es este un tema a investigar y está pensado como proyecto de grado.

CAPÍTULO 3. OBJETIVOS

3.1 **Objetivo general**

Presentar una metodología para el diseño, uso eficiente y óptimo, de los diferentes tipos de recursos espaciales y temporales, en la solución de problemas específicos en sistemas aritméticos digitales secuenciales y combinaciones, teniendo en cuenta sus aspectos de complejidad.

3.2 **Objetivos específicos**

- Indagar y conocer la teoría relevante y aplicable al diseño de circuitos digitales aritméticos, teniendo presente la evaluación de sus aspectos de complejidad circuital (temporal y espacial).

- Determinar y evaluar los diferentes métodos actuales en el diseño de circuitos aritméticos digitales y evaluar su complejidad según el tipo de recurso.

- Presentar una metodología y pautas de diseño, que permitan la implementación eficiente y óptima de los recursos temporales en el diseño de circuitos aritméticos digitales.

- Mostrar ejemplos prácticos de diseño y de evaluación de la metodología presentada.

PARTE II.

MARCO TEÓRICO

CAPÍTULO 4. LENGUAJES FORMALES

Una regla es una expresión de la forma $\alpha \rightarrow \beta$, donde α y β son una cadena de símbolos en las cuales se encuentran elementos del alfabeto y constantes, como símbolos y obtienen el nombre de variables. Y una gramática es un conjunto de reglas. Una forma de verlo más claramente es con el EJEMPLO 4.1, donde la siguiente gramática el cual produce un subconjunto del español:

1. $\langle frase \rangle \rightarrow \langle sujeto \rangle \langle predicado \rangle$
2. $\langle sujeto \rangle \rightarrow \langle artículo \rangle \langle sustantivo \rangle$
3. $\langle artículo \rangle \rightarrow la|los$
4. $\langle sustantivo \rangle \rightarrow niña|insectos$
5. $\langle predicado \rangle \rightarrow \langle verbo \rangle$
6. $\langle verbo \rangle \rightarrow sonríe|invaden$

En la gramática las variables son $\langle frase \rangle$, $\langle sujeto \rangle$, $\langle artículo \rangle$, $\langle sustantivo \rangle$, $\langle predicado \rangle$ y $\langle verbo \rangle$, y la variables son $la, los, niña$ e $insectos$. La variable $\langle frase \rangle$ es considerada el símbolo inicial.

Al aplicar una gramática se inicia con una variable, el cual se llama símbolo inicial, y se aplican iterativamente las reglas gramaticales, hasta que no haya variables en la palabra. En ese momento la palabra resultante es creada por la gramática, o en forma equivalente, la palabra resultante es parte del lenguaje de dicha gramática.

Se puede usar la gramática presentada anteriormente, para generar la frase "la niña sonríe".

$$\langle frase \rangle \rightarrow \langle sujeto \rangle \langle predicado \rangle$$

Reemplazando $\langle sujeto \rangle$ por $\langle articulo \rangle \langle sustantivo \rangle$ se obtiene

$$\langle frase \rangle \rightarrow \langle artículo \rangle \langle sustantivo \rangle \langle predicado \rangle$$

Reemplazando $\langle artículo \rangle$ por la se consigue

$$\langle frase \rangle \rightarrow la \ \langle sustantivo \rangle \langle predicado \rangle$$

Reemplazando $\langle sustantivo \rangle$ por $niña$ se tiene

$$\langle frase \rangle \rightarrow la\ niña \langle verbo \rangle$$

Se reemplaza $\langle verbo \rangle$ por $sonríe$ y finalmente se obtiene

$$\langle frase \rangle \rightarrow la\ niña\ sonríe$$

CAPÍTULO 5. JERARQUÍA DE CHOMSKY

A finales de la década de los cincuenta, el lingüista N. Chomsky, inició el estudio de las gramáticas formales, el cual fue el primero en proponer las gramáticas independientes del contexto, como un método de descripción de los lenguajes naturales y fue quien propuso una jerarquía de lenguajes, que relaciona a diferentes clases, en el cual las clases más complejas incluyen a las más simples, estas relaciones se definen en función de una complejidad estructural creciente [4]. Otro de sus principales hallazgos fue la demostración donde se puede construir modelos matemáticos, cuyas propiedades son un reflejo directo del grado de complejidad estructural de los lenguajes que se ajustan a los modelos, también señaló que estos modelos son de dos tipos: los autómatas y las gramáticas.

Se llama "clases de lenguajes" a los conjuntos de lenguajes que comparten ciertas propiedades, ver Figura 5.1:

- Los "Lenguajes Regulares" (T3), es la clase más pequeña, que incluyen a los lenguajes más simples, uno de ellos es el conjunto de todos los números binarios.
- Los "Lenguajes independientes de Contexto" (T2), que incluyen a los lenguajes regulares. Como son la mayoría de los lenguajes de programación.
- Los "Lenguajes Sensibles al Contexto" (T1), son moderadamente sensibles al contexto y preferible para las lenguas naturales.
- Los "Lenguajes Recursivamente Enumerables" (T_{rc}), que incluyen a los Libres de Contexto y por lo tanto a los Regulares.

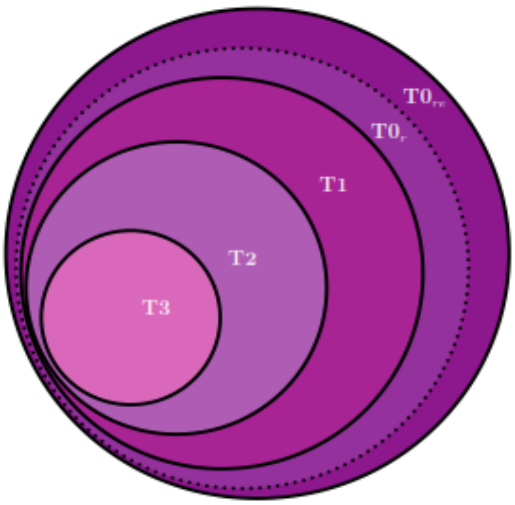


Figura 5.1 Jerarquía de Chomsky.

Todas estas clases de lenguajes son representables de manera finita, las cuales son más prácticas e ideales de ver como cadenas de caracteres [5].

5.1 LENGUAJES REGULARES

Los lenguajes regulares son el lenguaje básico de la Jerarquía de Chomsky y se les da ese nombre debido a que las palabras contienen repeticiones de los mismos componentes como por ejemplo $L_1 = \{ab, abab, ababab, abababab, ababababab, \dots\}$, en donde las palabras de L_1 son repeticiones de ab. Hasta ahora, se han descrito lenguajes más simples de los descritos por Chomsky. *"Los lenguajes regulares tienen ciertas propiedades de la fonología y quizás la morfología de las lenguas naturales son moldeables con lenguajes de complejidad estructural"* [5]. Dichos lenguajes tienden a caracterizarse porque las cadenas son lineales, de tal modo, que al hacer uso de un símbolo u otro en un campo determinado de dicha cadena depende exclusivamente del símbolo que lo precede.

5.1.1 Autómatas de estados finitos (AEF)

Los Autómatas de Estados Finitos (AEF), son un modelo computacional asociado a los sistemas regulares, es un modelo simple, el cual se puede imaginar como una unidad de control asociada a un cabezal de lectura, que está conectado a una cinta dividida en celdas y en dicha celda hay un símbolo de la cadena al que se va a analizar, más «#», el cual indica el final de la cadena. La cabeza sólo se desplaza hacia la derecha y va leyendo los símbolos sucesivamente hasta que cumple el ciclo, el cual es de un número finito de estados. En la **Figura 5.1.1,** se puede observar una representación de cómo sería gráficamente el AEF, donde la flecha doble indica el sentido en el que se mueve el cabezal.

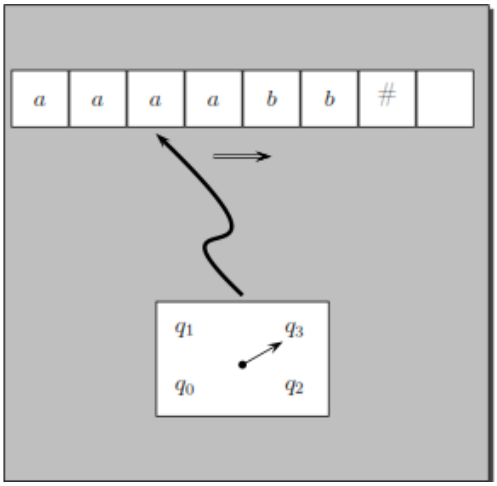


Figura 5.1.1 Autómata de estados finitos

A este modelo básico se pueden hacer diferentes mejoras no muy radicales, como permitir que el cabezal se desplace en ambas direcciones, que no sólo pueda leer, sino también borrar o escribir en la cinta, la unidad de control pueda guardar lo leído en una pila de memoria. Al hacerle dichas mejoras, ya el autómata será capaz de reconocer lenguajes estructuralmente más complejos. Hay que tener en cuenta, cualquier autómata construido con dichas especificaciones sólo podrá reconocer lenguajes regulares, ya que su capacidad para hacerlo depende de cómo se han definido los estados y sus posibles transiciones.

5.1.2 Gramáticas formales

Una gramática es el conjunto de reglas para formar correctamente frases en un lenguaje deseado. Una de las reglas es la expresión $\alpha \rightarrow \beta$, en donde α y β, son una cadena de símbolos

en la cual se pueden presentar elementos del alfabeto, constantes y variables. Por ejemplo, una regla gramatical puede ser $X \rightarrow aX$, aplicando la regla $\alpha \rightarrow \beta$ a una palabra $p\alpha d$ produce la palabra $p\beta d$, por lo tanto, las reglas gramaticales se pueden interpretar como un reemplazo. Otro ejemplo, teniendo $aaXb$ como cadena de símbolos, y aplicándoles la regla $X \rightarrow aX$, se tiene como resultado $aaaXb$

5.1.3 Gramáticas regulares

La gramática regular se percibe como un sistema capaz de generar lenguajes, éste es como el programa que implementa el autómata, en el cual con una mínima modificación se puede convertir en un dispositivo generador de lenguajes, de tal modo, que el cabezal, en vez de ir leyendo los símbolos de la cinta, va a ir escribiendo sobre las celdas en blanco.
El interés principal es por las gramáticas, donde las reglas de la forma $A \rightarrow aB$ o de la forma $A \rightarrow a$, en donde A y B son variables y a es un carácter terminal. Son llamadas gramáticas a las siguientes reglas:

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

Lo principal de una gramática, es que se parte de una variable con el nombre de símbolo inicial y se aplican iterativamente las reglas gramaticales, hasta que se terminen las variables en la palabra; a dicho resultado se dice, que es generada por la gramática.

Por definición una gramática G es una cuádrupla $G = (V, \Sigma, S, P)$, donde V, es un alfabeto de símbolos no terminales (variables), Σ es un alfabeto de símbolos terminales (constantes) y $V \cap \Sigma = 0, S \in V$ es el símbolo inicial, y P es un conjunto finito de reglas. Las reglas de P son siempre de la forma $x \rightarrow y$, tal que x e y son cadenas sobre Σ ∪ V y $x \neq \varepsilon$ [5]. Las aplicaciones de una gramática se formalizan de la siguientes formas:

- Una cadena pXd deriva de una cadena $p\alpha d$, se escribe como $pXd \Longrightarrow p\alpha d$, cuando hay como regla $X \rightarrow \alpha \in R$.
- Una cadena $\omega \in \Sigma^*$(formada exclusivamente por constantes), es derivable a partir de una gramática G, sí existe una secuencia de pasos derivados $S \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \alpha_3 \Longrightarrow \cdots \Longrightarrow \omega$.

Al momento de diseñar gramáticas regulares, se puede incurrir en los mismos errores del Automata Finito, como es producir palabras que no deberían, incompletas, palabras que no pertenecen al lenguaje a usar.

5.2 LENGUAJES INDEPENDIENTES DE CONTEXTO

Este lenguaje es importante, tanto desde el punto de vista teórico, por relacionar las Gramáticas Independientes de Contexto con los Autómatas de Pila y desde el punto de vista práctico, la mayoría de los lenguajes de programación están basados en él,

5.2.1 **Gramáticas independientes de contexto**

Las gramáticas independientes de contexto es una cuádrupla $G = (V, T, P, S)$, donde V y T son conjuntos variables y terminales respectivamente y $V \cap T = 0$. P, son un conjunto finito de producciones o reglas y todos los miembros del conjunto son de la forma $A \rightarrow \alpha$, donde A es una variable y α es una cadena de símbolos en $(V \cup T)^*$. S , es la variable denominada símbolo inicial.

Las gramáticas independientes de contexto son flexibles en algunas restricciones sobre el formato de reglas en comparación con las gramáticas regulares. Una de sus propiedades es que puede distinguir a los lenguajes regulares de los independientes de contexto, la linealidad. Las gramáticas lineales se caracterizan porque a la derecha de flecha, un único terminal es seguido por una cadena de símbolos terminales. Ahora a la derecha de la flecha se puede tener cadenas que combinen en cualquier orden terminales y no terminales, de tal modo, que la dependencia entre terminales ya no es estrictamente lineal.

5.2.2 **Autómatas de pila**

Los autómatas de pila, son la clase de autómatas equivalentes a las gramáticas independientes del contexto. Al tomar como referencia un AEF, el autómata de pila posee los mismos tres elementos básicos los cuales son: la unidad de control, el cabezal de lectura con desplazamiento hacia la derecha y una cinta donde se disponen en celdas los símbolos de la cadena, que el autómata debe reconocer. El autómata de pila además de esos tres elementos incorpora dos más: una cinta auxiliar o pila y un cabezal inicial, el cual es capaz de leer, escribir y borrar símbolos de la pila. Como se observa en la **Figura 5.2.2.1**

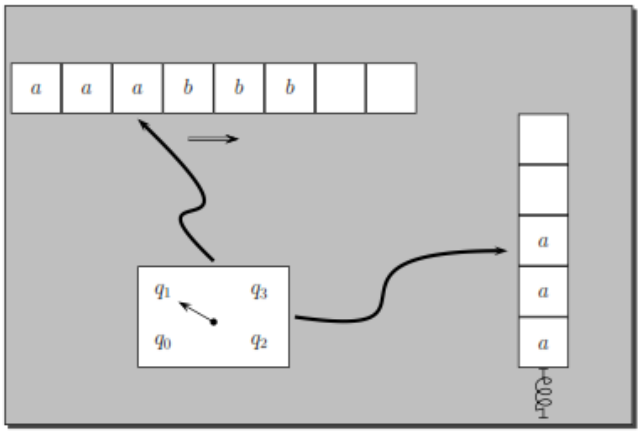


Figura 5.2.2.1 Un autómata de pila.

Como se observa en la **Figura 5.2.2.1**, el cabezal de lectura apunta a la cadena, mientras que el cabezal adicional va escribiendo lo que lee el cabezal de lectura y lo va escribiendo en la celda de la pila. Como se muestra, el cabezal va llenando las celdas de abajo hacia arriba, de tal modo que el primer símbolo está por debajo de los demás, el cual corresponde con el primer símbolo de la cadena de la cinta.

5.3 **GRAMÁTICAS Y LA JERARQUÍA DE CHOMSKY**

Las reglas gramaticales se restringen de manera que se acomoden a los patrones predeterminados. N. Chomsky propuso varias reglas, las cuales son estándares a las que se asocian con varias clases de lenguaje, de forma tal, que los lenguajes más básicos están incluidos en los más complejos. De esa forma se obtiene las clases de gramáticas, agrupadas a las familias de lenguaje:

1. Gramáticas Regulares (T3): las reglas son de la forma $A \to aB$ o $A \to a$, donde A y B son variables y a es constante.
2. Gramáticas Independientes de Contexto (T2): las reglas son de la forma $X \to \alpha$, donde X es una variable y α es una cadena, que puede contener tanto variables como constantes.
3. Gramáticas Sensibles al Contexto (T1): las reglas son de la forma $\alpha AB \to \alpha\Gamma\beta$, donde A y B son variables y α, β y Γ son cadenas las cuales pueden contener variables y constantes.
4. Gramáticas no Restringidas (T0): las reglas son de la forma $\alpha \to \beta$, donde α no puede ser vacío, que generan los lenguajes recursivamente enumerables.

Ahora, después de mostrar dos de las cuatro familias más importantes del lenguaje, ambas familias definen un espacio que, desde el punto de vista de la complejidad computacional, marca una frontera entre los procedimientos computacionales eficientes y aquellos que no lo son. Este es uno de los motivos del porqué todos los lenguajes de programación son construidos de tal modo, que su complejidad estructural nunca exceda de un sistema independiente de contexto.

CAPÍTULO 6. MÁQUINA DE TURING

En 1928 David Hilbert, un matemático, en el VIII Congreso Internacional de Matemáticos, dio origen a tres cuestiones, en el preguntó sí las matemáticas son completas, coherentes y decidibles. En 1931 Kurt Gödel publicó la demostración de sus dos teoremas, en donde resolvió la primera de las tres preguntas, en los cuales demostró que las matemáticas no son completas. En la década de los 30 entre 1935 y 1937, tanto Turing, desde Reino Unido y Alonzo Church de EE. UU, simultáneamente e independiente demuestran, que la tercera pregunta de Hilbert acerca del problema de decisión no tiene solución, al ver esto tanto Turing, como Church, decidieron dar un concepto formal de lo que es un algoritmo. Church decidió demostrarlo con base al cálculo de lambda, el cual él mismo lo desarrolló. Turing por su parte creo un modelo basado en la computación de una máquina, lo que con el tiempo terminaría llamándose la Máquina de Turing (MT). En su trabajo demostró, que ningún algoritmo distinguía entre dos expresiones en el cálculo de lambda; Turing demostró que una MT de nombre *D* y una cadena *ω* es imposible de determinar si *D* se detiene en un numero finito de pasos haciendo uso de *ω* como entrada, por lo cual se le conoció con problema de la parada. "*La MT define los límites de aquello que es computable y no existe un dispositivo más potente que este; superar los límites que impone la MT es superar los límites de la computabilidad."* [5](pag.75). En la tesis de Church-Turing dice que una función es algorítmicamente computable, sí y solo sí, es computable en la MT. La MT, es un modelo matemático, el cual representa una máquina teórica, a pesar de su simplicidad tiene el mismo poder computacional que una computadora de propósito general.

Anteriormente se explicó que al tener un autómata básico finito y añadirle una pila de almacenamiento, se aumenta la capacidad de cálculo; la MT, como los autómatas, tienen un control finito, un cabezal y una cinta con la que leer los símbolos. La cinta es de longitud infinita y tiene un movimiento hacia la derecha. Por otro lado, en la MT, el cabezal es de lectura y escritura, por lo tanto, la cinta puede cambiar el curso de ejecución, además, el cabezal se puede mover bidireccionalmente, por lo tanto, la MT, dependiendo del símbolo leído y del estado actual, el autómata cambia de estado o escribe un símbolo en la celda analizada o desplaza el cabezal de lectura/escritura a la izquierda o derecha en una posición. Dicho lo anterior se puede decir, que puede pasar repetidas veces por el mismo segmento de la cinta como ve en la **Figura 6.1Error! Reference source not found.**

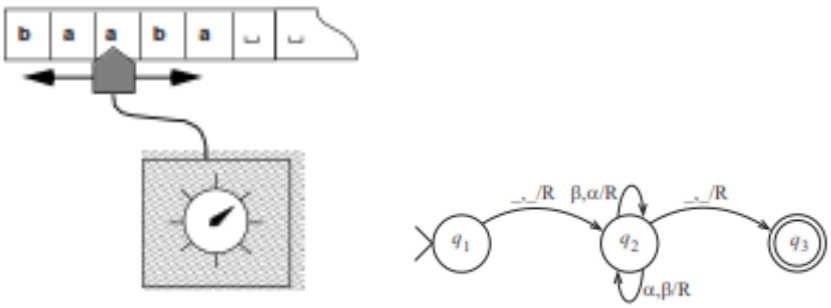


Figura 6.1 Maquina de Turing.

La MT funciona de la siguiente manera:

1. Lee el carácter en la cinta.
2. Hace una transición de estado.
3. Realiza una acción en la cinta; la cual puede ser:
    - Escribir el símbolo en la cinta.
    - Mover el cabezal a la derecha o a la izquierda.

La cinta únicamente puede hacer sólo una de las dos acciones, lo cual quiere decir, que son excluyentes.

La palabra de entrada en la MT, está escrita inicialmente en la cinta, lo que es habitual en los autómatas, pero iniciando desde la segunda posición de la cinta, siendo la primera celda un carácter en blanco. Como la cinta es infinita, inicialmente toda la parte de la cinta a la derecha de la palabra de entrada está llena del carácter blanco.
Se dice que en la MT, llega al "final de un cálculo", cuando se alcanza un estado especial llamado *halt (h,)* en el control finito, como resultado de una transición. Al llegar a *h* se detiene la operación de la MT, y se acepta la palabra de entrada, en la MT, no hay estados finales. Esto quiere decir que el *halt* es el único estado final, pero, también detiene la ejecución. Cuando se desea que una palabra no sea aceptada por la MT, se debe evitar que llegue a *halt*, haciendo que la máquina caiga en un ciclo infinito.
Al diseñar una MT, que acepte un lenguaje, en realidad se diseña un autómata finito, que controla el cabezal y la cinta, éste es un autómata con salida de Mealy (en el cual la salida depende del estado en que se encuentra el autómata).
Por definición la MT, es una séptupla $M = (Q, \Sigma, \Gamma, \delta, B, q_0, F)$, donde:
Q= conjunto finito de estados;
Σ= alfabeto de entrada;
Γ= alfabeto de la cinta, donde $\Sigma \subset \Gamma$ y $Q \cap \Gamma = \emptyset$;
$\delta \in (Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathbb{I}, \mathbb{D}\})$ Es la función de transición;
B= símbolo blanco, tal que $B \in \Gamma$ y $B \notin \Sigma$;
$q_0 \in Q$ es el estado inicial;
$F \subseteq Q$ es el conjunto de estados finales;

El alfabeto de la cinta Γ, incluye el alfabeto de entrada, más el símbolo blanco de B; y $Q \cap \Gamma = \emptyset$, impide que un estado pueda ser interpretado como símbolo. Las transiciones de la MT, se definen por δ, que toma valores pares ordenados obtenidos por un estado $q \in Q$, el cuál es el estado en el que se encuentra la MT, en el momento y el símbolo $X \in \Gamma$, que es el estado que está leyendo el cabezal. δ es una tripleta (p, Y, D), donde $p \in Q$ es el nuevo estado, $Y \in \Gamma$ es el símbolo que la MT, debe escribir en la celdilla, que está leyendo el cabezal y D es una bidireccional $\mathbb{I}$ o $\mathbb{D}$, que es el que indica si el cabezal se desplaza a la derecha o a la izquierda. Por lo tanto, esta MT es determinista, puesto que para cada par ordenado de estados y símbolos, δ se le asigna una tripleta, aunque no tiene por qué estar definido para todo su dominio, por eso puede haber pares de estados y símbolos para los que no exista una acción definitiva.

6.1 RESTRICCIONES DE LA MÁQUINA DE TURING

La MT se le puede poner restricciones a la definición sin que afecte la potencia computacional y las restricciones se aplicaran sucesivamente a:

1. El alfabeto.
2. La estructura de la cinta.
3. La capacidad de la máquina para realizar diferentes operaciones (escritura, desplazamiento o cambio de estado) en un solo cambio.

6.2 LA MÁQUINA DE TURING CON EL ALFABETO BINARIO

La MT, siempre realiza la misma tarea. Con el alfabeto binario $\Gamma = \{0,1,b\}$, donde b es el espacio en blanco $b \in \Gamma$, pero $b \notin \Sigma$. Para conseguirlo, es necesario codificar en binario los caracteres del alfabeto original. Cada transición original se desglosará en varias transiciones de la máquina con el alfabeto binario con el incremento de estados. La MT, en estos casos, el funcionamiento es de la siguiente manera:

1. Analiza los *n* caracteres que representa a un símbolo del alfabeto original.
2. Ya una vez reconocido el símbolo, la MT (definida sobre un alfabeto binario), debe retroceder como máximo *n* posiciones, con el fin de modificar la cadena de longitud *n* que acaba de realizar.
3. Por último, el cabezal debe desplazarse hasta llegar a la posición adecuada.

EJEMPLO 6.2.1

Se va a diseñar una MT, para ver cómo se comporta con una entrada típica, la cual aceptará el lenguaje $\{0^n1^n|n \geq 1\}$. Inicialmente, se proporciona a la cinta una secuencia finita de ceros y unos, seguida por secuencias infinitas de espacios en blanco; la MT cambiará primero un 0 por X y luego un 1 por Y, hasta que haya cambiado todos los 0 y 1.

Comenzando por el extremo izquierdo de la entrada, se cambia sucesivamente un 0 por una X y se mueve hacia la derecha pasando por encima de todos ceros y letras Y que lea, hasta cambiar el 1 por una Y, y se mueve hacia la izquierda pasando sobre todas las letras Y, y ceros hasta encontrar una X. en esta situación, busca un 0 colocado inmediatamente a la derecha y si lo encuentra lo cambia por una X y repite el proceso cambiando el 1 correspondiente por una Y.

Si la entrada no es 0 o 1, la MT terminará no haciendo el siguiente movimiento y se detendrá sin aceptar. Sin embargo, si termina cambiando todos los ceros y unos por X y Y respectivamente, se determinará que la entrada era de la forma 0^n1^n y acepta. La especificación formal de la máquina de Turing M es:

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_4\})$$

Donde δ es especifica en la **Tabla 6.2-1**

	Símbolo				
Estado	0	1	X	Y	B
$\boldsymbol{q_0}$	(q_1, X, R)	-	-	(q_3, Y, R)	-
$\boldsymbol{q_1}$	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
$\boldsymbol{q_2}$	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
$\boldsymbol{q_3}$	-	-	-	(q_3, Y, R)	(q_4, B, R)
$\boldsymbol{q_4}$	-	-	-	-	-

Tabla 6.2-1.Maquina de Turing que acepta $\{\boldsymbol{0^n1^n|n \geq 1}\}$

Mientras M, realiza las operaciones anteriores, la parte de la cinta que ya ha sido recorrida por el cabezal de la misma, corresponderá siempre a la secuencia de símbolos descrita por la expresión regular X*0*Y*1*. Esto quiere decir, que habrá ceros que fueron sustituidos por X, seguidos de ceros, que todavía no lo han sido. Luego se encontrarán algunos unos que han sido sustituidos por Y, y unos que no lo han sido.

El estado $q_0$es el estado inicial y *M* entra en el estado q_0 cada vez que vuelve al cero más a la izquierda que queda. Si *M* está en el estado q_0 y se señala un 0, la regla de la esquina superior izquierda de la **Tabla 6.2-1** indica que *M* tiene que pasar al estado q_1, cambiar el 0 por una X y moverse a la derecha. Una vez que está en el estado q_1, *M* se mueve hacia la derecha saltándose todos los ceros y las Y que encuentra en la cinta, permaneciendo en el estado q_1. Si *M* ve una X o una B, se detiene. Sin embargo, si *M* ve un 1 estando en q_1, cambia dicho uno por una Y, pasa al estado q_2 y comienza a moverse hacia la izquierda.

En el estado q_2, *M* se mueve hacia la izquierda pasando por encima de los ceros y las Y, permaneciendo en el estado q_2. Cuando M, alcanza a la X, que está más a la derecha, la cual marca el extremo derecho del bloque de ceros, que ya han sido cambiados por X, *M* vuelve al estado q_0 y se mueve hacia la derecha. Hay dos casos:

1. Si ahora *M* ve un 0, entonces repite el ciclo de sustituciones que se acabó de describir.
2. Si *M* ve una Y, entonces es que ha cambiado todos los ceros por X, si todos los unos han sido cambiados por Y, entonces la entra era de la forma 0^n1^n, y *M* acepta. Por lo tanto, *M* pasa al estado q_3, y comienza a moverse hacia la derecha, pasando por encima de las Y. Si el primer símbolo distinto de una Y, que *M* encuentra es un espacio en blanco, entonces existirá el mismo número de ceros que de unos, eso quiere decir que *M* entra en el estado q_4 y acepta. Por otro lado, si *M* encuentra otro 1, entonces es que hay una gran cantidad de unos, por lo que *M* deja de operar sin aceptar. Si encuentra un 0, entonces la entrada era de forma errónea y *M* también se detiene.

Considerando que la entrada es 0011, *M* se encuentra en el estado q_0, el cual es 0, es decir, la configuración inicial de *M* es $q_0 0011$. Y su secuencia completa de movimiento sería:

$$q_0 0011 \vdash Xq_1 011 \vdash Xq_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1 \vdash$$
$$Xq_0 0Y1 \vdash XXq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY \vdash$$
$$XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 B \vdash XXYYBq_4 B$$

Como se observa *M* no realiza ningún movimiento sí el símbolo de la entrada al que señala es B; por lo tanto, *M* deja de funcionar y no acepta la entrada.

6.3 LA MÁQUINA DE TURING PARA CÁLCULOS DE FUNCIONES

La MT, puede ser usada para calcular resultados u operaciones a partir de la entrada. Cuando el contenido de la cinta llega al *halt*, en lugar, de considerarlo como basura se considera como el resultado calculado. Para interpretar sin problema el contenido final de la cinta como resultado, es necesario que cumpla con un formato estricto, que se puede observar a continuación:

- La palabra de salida no puede tener ningún carácter blanco.
- La palabra de salida comienza en el segundo carácter de la cinta, teniendo a la izquierda un blanco y a la derecha una infinidad de blancos.
- El cabezal debe estar posicionado en el primer blanco a la derecha de la palabra de salida.

6.4 LÍMITES DE LA MÁQUINA DE TURING

Existen problemas que no se pueden resolver, como es una secuencia determinista de operaciones elementales, lo cual es lo esencial de las MT. Estos problemas son llamados *algorítmicamente irresolubles*. También están los problemas de palabras (*Word problem*). Se dice que un lenguaje es decidible, si hay una MT para decidir el problema de pertenencia de palabras. Otro de los problemas es la equivalencia de las gramáticas libres independiente de contexto, también está la ambigüedad de las GLC y el problema de la pertenencia de palabras para gramáticas sin restricciones.

Problema de paro de MT

Se considera un problema irresoluble, el cual históricamente tuvo mucha importancia, ya que fue el primer problema que se probó irresoluble. Una vez que se cuenta dicho problema, la forma de demostrar que también es irresoluble, es probar que estos pueden ser reducidos al problema de referencia y el primer problema irresoluble es el del paro de la MT.

El problema de paro consiste en determinar algorítmicamente si la MT va a parar o no, cuando analiza la palabra de entrada. Cabe resaltar que cuando una MT analiza el comportamiento de otra, se necesita que la última sea dada como la entrada de la primera; para recurrir a esto se necesita codificar la MT que se va a analizar. Una forma de codificar una MT es considerar una cadena de símbolos de su representación.

6.5 LA MÁQUINA DE TURING EN LA JERARQUÍA DE CHOMSKY

Las MT, no son capaces de aceptar todos los lenguajes posibles. Sin embargo, ésto se puede observar simplemente a partir de la enumerabilidad de las MT. Puesto que las MT son cuádruplos $(\mathrm{K}, \Sigma, \delta, s)$ (debido a eso los elementos de un producto cartesiano), como es enumerable cada uno de los componentes, necesariamente el cuádruplo es también enumerable. Por consiguiente:

- Los conjuntos de los posibles estados son enumerables y se estandarizan los nombres de los entados por $q_0, q_1, q_2, q_3,$ etc…por lo tanto, no altera ningún aspecto en la actividad de la MT.
- Igualmente, el alfabeto estándar $\sigma_0, \sigma_1, \sigma_2, \sigma_3,$ etc…, puede codificar cualquier alfabeto en particular. Lo mismo los alfabetos son enumerables.
- La transición es parte de otros productos cartesianos de caracteres y estados, lo que lleva a que sea también enumerable.
- Los estados iniciales son enumerables y siguiendo la estandarización del primer punto expresado anteriormente.

Como se habló anteriormente la MT, al ser enumerable, no puede ser un elemento de 2^{Σ^+}con una MT distinta, por eso hay lenguajes que no tienen una MT, que los acepte. Este resultado no ayuda para saber que lenguaje no son aceptados por ninguna MT.

Recordando la Jerarquía de Chomsky, donde clasifica los lenguajes por categorías y éstas están asociados a los distintos tipos de máquinas, las cuales dependen de las diferentes categorías de lenguaje.

Tipo de autómata	**Lenguaje que procesa**	**Gramática que lo genera**
Autómatas finitos	Lenguajes Regulares	Gramáticas Regulares
Autómatas de pila	Lenguajes Independientes de Contexto	Gramáticas Independientes de Contexto
Autómatas linealmente acotados	Lenguajes Sensible al Contexto	Gramáticas Sensible al Contexto
Máquina de Turing decidiendo	Lenguajes Recursivos	
Máquina de Turing aceptado	Lenguajes Recursivos Enumerables	Gramáticas no Restringidas

Tabla 6.5-1. Clases de lenguajes que pueden ser decididos por un MT.

CAPÍTULO 7. COMPUTABILIDAD Y COMPLEJIDAD.

7.1 **COMPUTABILIDAD**

La teoría de la computabilidad principalmente se centra en estudiar y clasificar problemas que se solucionan con un algoritmo; si son algorítmicamente solucionables o no solucionables. O si son equivalentes a una Máquina de Turing. Cuando se empezó a hablar de la teoría de la computabilidad, se hizo por la necesidad de resolver una de las preguntas que planteó previamente Gödel, en la cual Turing y su Máquina de Turing le dio solución a una de ellas. A raíz de la MT, se plantearon las siguientes preguntas:

- ¿Qué problemas tiene la capacidad de resolver una Maquina de Turing?
- ¿Qué formulismos son equivalentes a la MT?
- ¿Qué problemas requieren maquinas con mayor capacidad?

7.1.1 **¿Qué problemas tiene la capacidad de resolver una MT?**

Uno de los problemas que no se puede resolver algorítmicamente es uno indecidible, aunque se disponga de espacio y tiempo; uno de ellos sería el Entscheidungsproblem (problema de decisión) el cual Church y Turing demostraron que independientemente este problema es indecidible. Un segundo problema es el de parada, en el que determina que teniendo un programa y su entrada, decidir si dicho programa cumplirá su tarea para esa entrada o sí continuará indefinidamente. Si el número es uno computable y es uno real, que puede ser aproximado por un algoritmo de exactitud arbitraria, Turing demostró que casi ningún número es computable como por ejemplo la Constante de Chaitin, el cual no es computable, pero si está bien definida.

7.1.2 **¿Qué formulismos son equivalentes a la MT?**

Los lenguajes que son aceptados por una MT, son los que son generados por la gramática formal; el cálculo de Lambda es uno de ellos. Por lo tanto, las funciones que puedan ser computadas por el cálculo de Lambda son aquellas que también puedan ser computadas por la MT, aun cuando son desarrolladas por diferentes personas como se demostró en la Tesis Church-Turing. Los computadores basados en la arquitectura Von-Neumann tienen el mismo poder de expresión que la MT, sí se dispusiera de recursos ilimitados de tiempo y espacio. Por lo tanto, los lenguajes de programación tienen teóricamente como máximo el mismo poder de expresión que un programa para una MT.

7.1.3 **¿Qué problemas requieren una máquina con mayor capacidad?**

Una máquina oráculo que utiliza una caja negra tiene mayor capacidad, ya que puede calcular una función en particular, que no es calculable en la MT. La fuerza de cómputo de una máquina oráculo es descrita por el grado de Turing.

7.2 **COMPLEJIDAD**

En un sistema digital la teoría de la complejidad es de gran relevancia, ya que define en donde se halla la frontera que separa los problemas tratables de los intratables; por eso existen diferentes maneras de evaluar la dificultad del problema. Cuando se tiene un problema tratable

se pueden tener diferentes métodos, sobre los cuales normalmente, se determinan los parámetros relacionados con la eficiencia y la simplicidad del algoritmo. Al hacer mención de la eficiencia, se habla del uso eficiente de los recursos de los cuales dispone el algoritmo para la solución del problema, normalmente son espaciales y temporales. El recurso espacial habla acerca de la cantidad de uso de memoria requerida para dar solución a dicho problema; aunque puede ser de importancia, este recurso no es tan relevante como lo es el temporal. La simplicidad en un algoritmo es algo a destacar, puesto que facilita realizar un seguimiento versátil del mismo, también facilita la legitividad y mantenibilidad frente a otras soluciones del algoritmo.

7.2.1 **Recurso temporal**

El recurso temporal de un algoritmo se puede obtener de diferentes maneras; uno de los métodos es midiendo el tiempo, desde el punto de inicio del algoritmo, hasta obtener la respuesta de solución del problema. Sin embargo, aunque esa respuesta sea útil, no representa importancia al momento de hacer una comparación con otros algoritmos con recursos diferentes. Un segundo método para calcular este recurso es aproximando el valor obtenido, por encima o por debajo del valor real y en función del número total de pasos del algoritmo; este método realiza una evaluación a priori e independiente del sistema donde sea ejecutado, por lo tanto, este método es de mayor interés. El recurso temporal a priori, se basa en la medición del número total de operaciones elementales implementados por el algoritmo; al mencionar las operaciones elementales se hace referencia a aquellas realizadas por el algoritmo en un tiempo finito, el cual no es mayor a una constante dada, entre estas operaciones se tiene en cuenta todas las operaciones aritméticas básicas, las funciones de comparación; las cuales se toman un tiempo de ejecución $1OE$.

Con los siguientes tres ejemplos se muestran tres algoritmos diferentes, pero que van a realizar la misma tarea, la cual va a hacer el ordenamiento ascendente de un vector de entrada con números naturales entre 0 y 60.

a) El primer método es clásico y es llamado Ordenamiento por Recorrido Unidireccional y radica en recorrer todo el vector en una sola dirección e ir indagando sí el elemento en la posición i, es mayor que el elemento en la posición $i + 1$, en tal caso, se deben intercambiar ambos elementos; este proceso se debe hacer iterativamente hasta que el vector este ordenado. El código a continuación es la implementación y es desarrollado en Matlab:

```
01      function [SALIDA] = Unidireccional (ENTRADA)
02
03              for j = 1: (length (ENTRADA)-1)
04                      for i = 1: (length (ENTRADA)-1)
05                              if ENTRADA (i) > ENTRADA (i +1)
06                                      TEMP = ENTRADA (i);
07                                      ENTRADA (i) = ENTRADA (i +1);
08                                       ENTRADA (i +1) = TEMP;
09                               else ENTRADA=ENTRADA;
10                              end
11                      end
12              end
13
14      SALIDA = ENTRADA
```

El algoritmo entre las líneas 03 y 14 se desarrolla con el siguiente número de operaciones elementales:

03 $3OE$: tamaño del vector, una resta y una asignación a la variable i
04 $3OE$: tamaño del vector, una resta y una asignación a la variable j
05 $4OE$: una suma, dos accesos al vector y una comparación
06 $2OE$: un acceso a vector y una asignación
07 $4OE$: una suma, dos accesos al vector y una asignación
08 $3OE$: una asignación, una suma y un acceso al vector
09 $1OE$: una asignación
14 $1OE$: una asignación

En el caso medio, la mitad de las veces se efectúa los renglones 06 a 08 y la otra mitad el renglón 09. Un promedio de lo que realizan los renglones 06 a 09 es de $^{(9+1)}/_{2}$=5 veces en cada iteración por lo que el tiempo de iteración es:

$$T_1(n) = \left(3 + \sum_{j=1}^{n-1}\left(3 + \sum_{i=1}^{n-1}(4+5)\right)\right) + 1 = (n-1)(9n-6) + 4$$
$$= 9n^2 - 15n + 10$$

b) El siguiente algoritmo es llamado Ordenamiento por Recorrido Bidireccional, el cual optimiza el método anterior, ya que se aprovecha un recorrido por el vector para ir optimizando tanto de derecha a izquierda, como de izquierda a derecha y hace provecho de que sólo en la primera iteración ya ordena el primer y el último elemento, y así sucesivamente. El código es desarrollado en Matlab:

```
01  function [SALIDA] = Bidireccional (ENTRADA)
02
03              k =0;
04              LONGITUD= length (ENTRADA);
05
06              for j = 1: (LONGITUD/ 2)
07                      for i = 1: (LONGITUD-1-2*k)
08                              if ENTRADA (i +k) > ENTRADA (i +1+k)
09                                      TEMP = ENTRADA (i +k);
10                                      ENTRADA (i +k) = ENTRADA (i +1+k);
11                                              ENTRADA (i +1+k) = TEMP;
12                              else ENTRADA=ENTRADA;
13                      end
14
15                      if ENTRADA (LONGITUD+1-i-k) <ENTRADA (LONGITUD-i-k)
16                              TEMP = ENTRADA (LONGITUD+1-i-k);
17                              ENTRADA (LONGITUD+1-i-k) ENTRADA (LONGITUD-i-k);
18                              ENTRADA (LONGITUD-i-k) = TEMP;
19                      else ENTRADA=ENTRADA;
20                      end
21              end
22                       k = k + 1;
23      end
24
25  SALIDA = ENTRADA
```

Para este método el algoritmo se ejecuta desde la línea 03 hasta las 23, con las siguientes operaciones elementales en cada una de ellas:

03 $1OE$: una asignación
04 $2OE$: evaluar el tamaño del vector y una asignación
05 $2OE$: una resta y una asignación
06 $4OE$: tres operaciones básicas y una asignación
07 $6OE$: tres operaciones básicas, dos accesos al vector y una comparación
08 $3OE$: una operación básica, un acceso al vector y una asignación
09 $6OE$: tres operaciones básicas, dos accesos al vector y una asignación
10 $4OE$: dos operaciones básicas, un acceso al vector y una asignación
11 $1OE$: una asignación
13 $8OE$: cinco operaciones básicas, dos accesos al vector y una comparación
14 $5OE$: tres operaciones básicas, un acceso al vector y una asignación
15 $8OE$: cinco operaciones básicas, dos accesos al vector y una asignación
16 $4OE$: dos operaciones básicas, un acceso al vector y una asignación
17 $1OE$: una asignación
20 $2OE$: una operación básica y una asignación
23 $1OE$: una asignación

En el caso medio, la mitad de las veces se ejecutan los renglones 08 a 10 y 14 a 16 y la otra mitad los renglones 11 y 17. De los renglones 08 a 11 se ejecuta $^{(13+1)}/_{2}$ =7 veces en cada iteración y en los renglones 14 a 17 se ejecuta $^{(17+1)}/_{2}$ =9 veces en cada iteración. Por lo anterior el tiempo de ejecución es:

$$T_2(n) = 3 + \left(2 + \sum_{j=1}^{n/2}\left(4 + \sum_{i=1}^{(n-1)/2}(6+7+8+9)+2\right)\right) + 1 = \frac{15}{2}n^2 - \frac{9}{2}n + 6$$

c) Y por último el algoritmo Ordenamiento por Casilleros, consiste en separar inicialmente en clases o casillas los elementos con el objetivo de ordenarlos por cualquier método, con la suposición de que es más sencillo ordenar pocos elementos, que muchos. El código se desarrolló en Matlab:

```
01  function [SALIDA] = Casilleros (ENTRADA)
02
03  CASILLA1 = [];                  %Para elementos entre 0 y 20
04  CASILLA2 = [];                  %Para elementos entre 21 y 40
05  CASILLA3 = [];                          %Para elementos entre 41 y 60
06
07              for i = 1: length (ENTRADA)
08                      ELEMENTO = ENTRADA (i);
09                      if ELEMENTO < 21
10                              CASILLA1 = [CASILLA1 ELEMENTO];
11                      else if ELEMENTO < 41
12                              CASILLA2 = [CASILLA2 ELEMENTO];
13              else CASILLA3 = [CASILLA3 ELEMENTO];
14                      end
15              end
16
17              for j = 1: (length (CASILLA1)-1)
18                      for i = 1: (length (CASILLA1)-1)
19                              if CASILLA1 (i) > CASILLA1 (i +1)
20                                      TEMP = CASILLA1 (i);
21                                      CASILLA1 (i) = CASILLA1 (i +1);
```

```
22                                      CASILLA1 (i +1) = TEMP;
23                                  else CASILLA1=CASILLA1;
24                                  end
25                          end
26                  end
27
28                  for j = 1: (l e n g t h (CASILLA2)-1)
29                          for i = 1: (length (CASILLA2)-1)
30                                  if CASILLA2 (i) > CASILLA2 (i +1)
31                                          TEMP = CASILLA2 (i);
32                                          CASILLA2 (i) = CASILLA2 (i +1);
33                                          CASILLA2 (i +1) = TEMP;
34                                  else CASILLA2=CASILLA2;
35                                  end
36                           end
37                  end
38
39                  for j = 1: (length (CASILLA3)-1)
40                          for i = 1: (l e n g t h (CASILLA3)-1)
41                                  if CASILLA3 (i) > CASILLA3 (i +1)
42                                          TEMP = CASILLA3 (i);
43                                          CASILLA3 (i) = CASILLA3 (i +1);
44                                          CASILLA3 (i +1) = TEMP;
45                                  else CASILLA3=CASILLA3;
46                                  end
47                          end
48                  end
49
50  SALIDA = [CASILLA1 CASILLA2 CASILLA3]
```

En el tercer de los algoritmos se ejecuta desde los renglones 07 a 50 con el siguiente número de operaciones elementales en cada una de ella:

07 $2OE$: evalua el tamaño del vector y una asignación
08 $2OE$: un acceso al vector y una asignación
09 $1OE$: una comparación
10 $2OE$: una concatenación y una asignación
11 $1OE$: una comparación
12 $2OE$: una concatenación y una asignación
13 $2OE$: una concatenación y una asignación
17, 28,39 $3OE$: evaluar tamaño del vector, una operación básica y una asignación
18, 29,40 $3OE$: evaluar tamaño del vector, una operación básica y una asignación
19, 30,41 $4OE$: una operación básica, dos accesos al vector y una comparación
20, 31,42 $2OE$: un acceso al vector y una asignación
21, 32,43 $4OE$: una operación básica, dos accesos al vector y una asignación
22, 33,44 $3OE$: una operación básica, un acceso al vector y una asignación
23, 34,45 $1OE$: una asignación
50 $2OE$: una concatenación y una asignación

En el caso medio, la mitad de las veces se ejecutan los renglones 20 a 22, 31 a 33 y 42 a 44 y la otra mitad no se ejecutan. En promedio el interior de los dos ciclos for anidados se ejecutan $^{(9+1)}/_{2}$ =5 veces en cada iteración, por lo que el tiempo de ejecución es:

$$T_3(n) = 2 + \sum_{i=1}^{n-1}(2+3) + 3 * \left(3 + \sum_{j=1}^{\frac{n}{3}-1}\left(3 + \sum_{i=1}^{\frac{n}{3}-1}(4+5)\right)\right) + 2$$

$$= 3n^2 - 10n + 15$$

En la **Figura 7.2.1.1**, se puede ver la comparación del caso medio de cada uno de los tres algoritmos, en donde la línea azul representa el Ordenamiento por Recorrido Unidireccional, en cuál es el primer algoritmo, el verde es Ordenamiento por Recorrido Bidireccional el cuál es el segundo algoritmo y por último, el rojo es el Ordenamiento por Casilleros y es el tercer algoritmo.

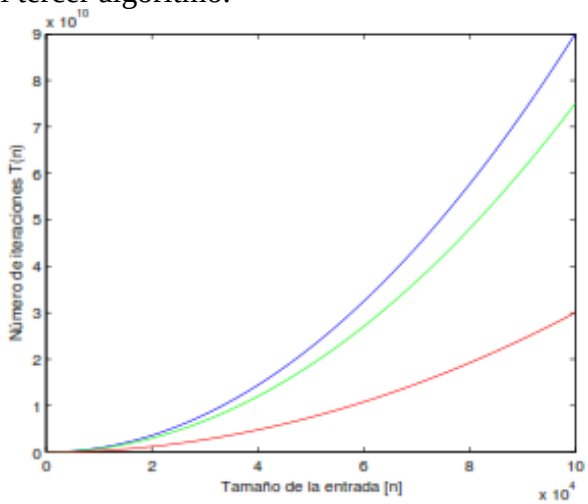


Figura 7.2.1.1 Comparación de recurso temporal de tres algoritmos

De la figura anterior se puede observar, inicialmente, el número de iteraciones solicitado por el tercero de los algoritmos es mucho más eficiente que los otros dos. Por otro lado, para entradas de gran cantidad, el segundo de los algoritmos tiene mejor desempeño y para entradas de tamaño pequeño, el ideal es el primer algoritmo; como se muestra en la **Figura 7.2.1.2** , el cual es una ampliación de **Figura 7.2.1.1**

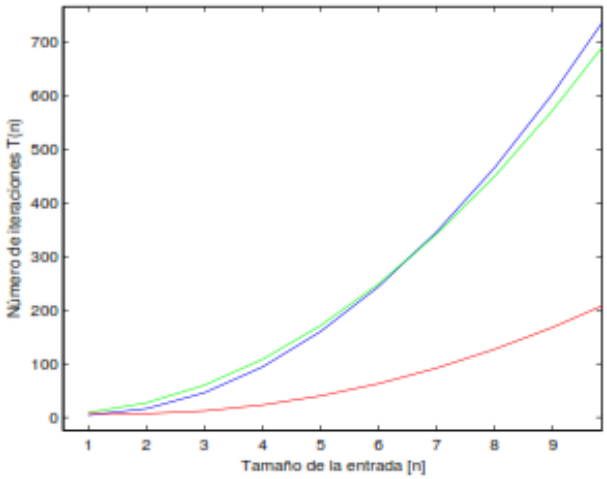


Figura 7.2.1.2 Comparación de algoritmos ante una entrada de tamaño pequeño

Medidas asintóticas

Las medidas asintóticas son usadas para comparar dos funciones, las cuales evalúan el tiempo de ejecución entre dos funciones con implementación diferente, o una sola implementación y diferentes funciones de referencia. A continuación, se habla de las más relevantes:

7.2.1.1. **Cota superior (**O**)**

Dada una función $f(n)$, se quiere encontrar las funciones $g(n)$, las cuales crecen tan rápido o al menos casi tanto como $f(n)$ y las cuales se llaman como cota superior de $f(n)$ y se nota por O(f). Si $g: \mathbb{N} \to [0,\infty)$, y se define:

$$O(g) = \{f: \mathbb{N} \to [0,\infty) | \exists c \in \mathbb{R}, c > 0, \exists n_0 \in \mathbb{N}: 0 \leq f(n) \leq cg(n) \forall n \geq n_0\}$$

Esto se puede interpretar como sí f y g son dos funciones con dominio y recorrido naturales, se dice que $f(n) = O(g(n))$, sí existen números c y n_0 tal que $f(n) \leq cg(n)$ para todo $n \geq n_0$.de lo contrario $f(n) \neq O(g(n))$. Si, además, $f(n) = O(g(n))$ y $g(n) = O(f(n))$ se dice que las dos funciones poseen la misma tasa de crecimiento, pero sí por el contrario $f(n) = O(g(n))$ y $g(n) \neq O(f(n))$, se dice que la función g crece mas rápido que la función f, esta explicación y su interpretación se ve claramente en la **Figura 1**

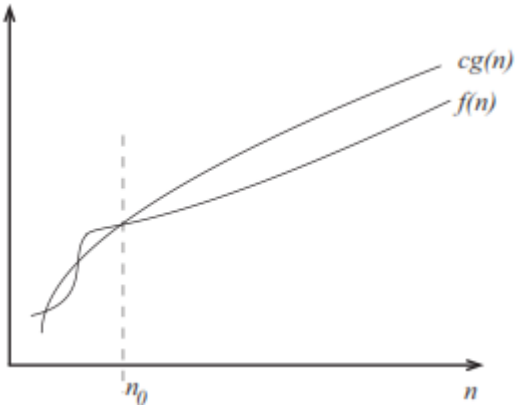


Figura 1 Notación asintótica superior

Para el caso promedio del tercer algoritmo de Ordenamiento por Casillas se dice que:

$$n^2 = O(3n^2 - 10n + 15)$$

Se cumple que:

$$\frac{n^2}{3n^2 - 10n + 15} = \frac{1}{3 - \frac{10}{n} + \frac{15}{n^2}} \overrightarrow{n \to \infty} \frac{1}{3}$$

El cual demuestra que existe un número n_0 tal, que para todo $n \geq n_0$ y se cumple

$$\frac{n^2}{3n^2 - 10n + 15} \leq 1$$

De lo anterior $3n^2 - 10n + 15 = O(n^2)$, dado que ambas funciones tienen la misma tasa de crecimiento. Por otro lado $n^3 \neq O(3n^2 - 10n + 15)$, muestra que:

$$\frac{n^3}{3n^2 - 10n + 15} = \frac{n}{3 - \frac{10}{n} + \frac{15}{n^2}} \overrightarrow{n \to \infty} \infty$$

Donde se concluye que n^3 tiene un crecimiento más rápido que $3n^2 - 10n + 15$

Un polinomio tiene la forma general

$$p(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_k n^k$$

Se dice que el grado del polinomio lo determina la cota superior de crecimiento ya que:

$$\frac{p(n)}{n^k} = \frac{a_0}{n^k} + \frac{a_1}{n^{k-1}} + \frac{a_2}{n^{k-2}} + \cdots + a_k \overrightarrow{n \to \infty} a_k$$

Finalmente, se demuestra que los tres algoritmos de ordenamiento de los ejemplos anteriores crecen todos igual de rápido como n^2.

COTA	NOMBRE
O(1)	Orden constante
O($logn$)	Orden logarítmico
O(n)	Orden inicial
O($nlogn$)	Orden lineal algorítmico
O(n^c)	Orden potencial
O(c^n)	Orden exponencial
O($n!$)	Orden factorial
O(n^n)	Orden potencial exponencial

Tabla 1-1: Cotas superiores comunes

7.2.1.2. **Cota inferior (Ω)**

Dada una función $f(n)$, se quiere encontrar funciones $g(n)$, que crezcan tan lentamente como $f(n)$, por lo que se denominan como cota inferior de $f(n)$ y notan por $\Omega(f)$. Si $g: \mathbb{N} \to [0, \infty)$ y se define:

$$\Omega(g) = \{f: \mathbb{N} \to [0, \infty) | \exists c \in \mathbb{R}, c > 0, \exists n_0 \in \mathbb{N}: 0 \leq cg(n) \leq f(n) \forall n \geq n_0\}$$

Esta expresión se interpreta como, sí f y g son dos funciones con dominio y recorrido en los naturales, se dice que $f(n) = \Omega(g(n))$, sí existen los números c y n_0 tal que $f(n) \geq cg(n)$ para todo $n \geq n_0$. De lo contrario $f(n) \neq \Omega(g(n))$. Si además, $f(n) = \Omega(g(n))$ y $g(n) = \Omega(f(n))$, entonces las dos funciones tienen la misma tasa de crecimiento, pero sí $f(n) = \Omega(g(n))$ y $g(n) \neq \Omega(f(n))$, la función g crece más lentamente, que la función f. En la **Figura 2** se puede ver la interpretación.

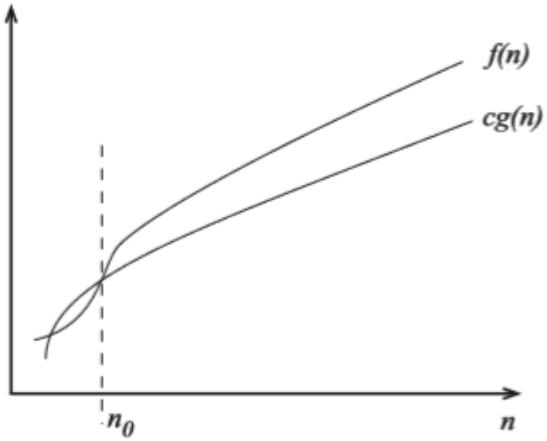


Figura 2 Notación asintótica inferior

7.2.1.3. **Cota exacta (Θ)**

Dada una función $f(n)$, se quiere encontrar funciones $g(n)$, que crezcan tan rápido y tan lentamente como $f(n)$ y es llamada cota exacta de $f(n)$ y se nota por $\Theta(f)$. Si $g: \mathbb{N} \to [0, \infty)$ y se define:

$$\Theta(g) = \{f: \mathbb{N} \to [0, \infty) | \exists c_1 \in \mathbb{R}, \exists c_2 \in \mathbb{R}, c_1 > 0, c_2 > 0, \exists n_0 \in \mathbb{N}: 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}$$

Esa expresión se puede interpretar como, sí f y g son dos funciones con dominio y recorrido en los naturales, se dice que $f(n) = \Theta(g(n))$, sí existen números c_1, c_2 y n_0 tal que $c_1 g(n) \leq f(n) \leq c_2 f(n)$ para todo $n \geq n_0$, eso quiere decir, que $f(n)$ está acotado inferiormente por $c_1 g(n)$ y superiormente por $c_2 g(n)$. De lo contrario $f(n) \neq \Theta\big(g(n)\big)$. En la **Figura 3** se puede ver la interpretación de forma gráfica.

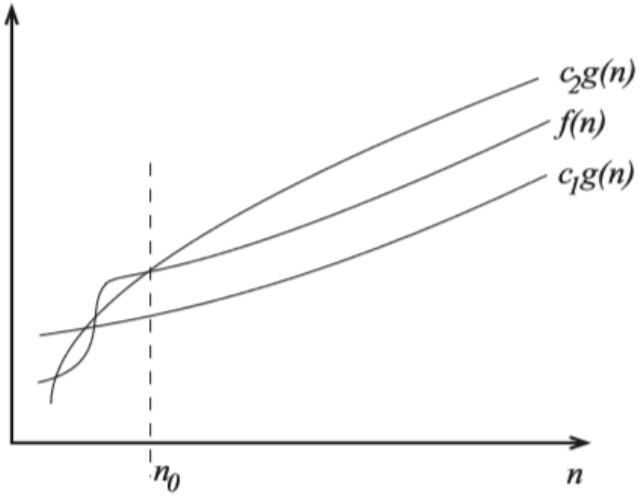


Figura 3 Notación asintótica exacta

La cota exacta también se puede ver de la siguiente forma:

$$\Theta(g) = O(g) \cap \Omega(g)$$

Conseguir establecer una cota ajustada a la función, no siempre es fácil, para conseguirlo se debe establecer previamente cuales van a ser las cotas superiores e inferiores. Estos parámetros son de gran importancia y muy útiles para conocer la complejidad de un determinado algoritmo. Esto no siempre es viable, dado que la teoría de la complejidad suele trabajar sólo con las cotas superiores. Lo cual es obvio, puesto que la cota superior determina la complejidad de un problema en el peor de los casos, es decir; por muy complejo que sea un problema nunca va a ser más complejo de lo que establece la cota superior.

7.2.2 **Complejidad P y NP**

Es estudiada por la teoría de complejidad, en la que parte de la teoría de computación se refiere a los recursos que requiere para el cálculo para así resolver problemas dados, tales como el tiempo y el espacio. Para este tipo de análisis se tiene en cuenta el recurso temporal, dado que los requerimientos son en términos del tiempo. Los modelos suponen que el sistema es determinista, dado que el estado actual y las variables de entrada sólo existe una posible acción que puede tomar; y secuencial, pues realiza una acción después de la otra. Los sistemas conservan estas características aun cuando el sistema tiene una función en paralelo. En la teoría de la complejidad computacional se cuestiona si P=NP.

7.2.2.1 **Problemas P**

Es el conjunto de problemas de decisión, que pueden ser resueltos en un sistema determinista, secuencias en un tiempo polinomial proporcional a los datos de entrada.

7.2.2.2. **Problemas NP**

Es el conjunto de problemas de decisión, donde las soluciones, ya sean afirmativas pueden ser verificados en un tiempo polinomial a partir de una alimentación de forma equivalente.

CAPÍTULO 8. CIRCUITOS Y SUS CARACTERÍSTICAS.

La complejidad en la electrónica se caracteriza por la complejidad en circuitos lógicos. Cuando se hace referencia a la complejidad en circuitos lógicos, normalmente se tiene en cuenta el tamaño y profundidad del sistema. Sí se compara un circuito con una Máquina de Turing, se observa que el cabezal de lectura de la MT, en el circuito vendría siendo los pasos de programa, el cual hace uso del recurso temporal y la cinta en la MT, vendría siendo el recurso espacial en el circuito. Cabe mencionar que sí una función es de un tamaño mayor a las entradas fija, entonces el tiempo requerido por la Máquina de Turing es largo. Un circuito generalmente está diseñado para funciones booleanas, los cuales son llamados circuitos lógicos, los circuitos que usan operaciones algebraicos son llamados circuitos algebraicos y aquellos que hacen uso de comparadores son llamados circuitos comparadores. La complejidad está relacionada únicamente a circuitos y la velocidad de respuesta del mismo, pero no necesariamente con la facilidad que en el lenguaje aceptado a usar; cada circuito diseñado es diferente pero con algunas características óptimas.

8.1**FAN-IN**

Es la cantidad de entradas que pueda tener una compuerta lógica dependiendo de la familia que sea, también está relacionada con la capacidad que dicha familia tenga para absorber la corriente de compuertas que están en la etapa anterior.

8.2**FAN-OUT**

Es la capacidad que tiene una compuerta lógica de alimentar y controlar varias compuertas (de una misma familia) desde su salida sin exceder la carga, según sea las especificaciones.

8.3**CAPAS**

Es el número de compuertas lógicas por las que pasa una señal desde la entrada hasta la salida

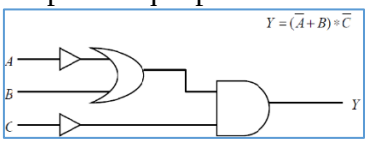


Figura 8.3 Ejemplo de capas

En el ejemplo en la **Figura 8.3** la entrada *A* con respecto a la salida *Y* tiene 3 capas y es la más crítica, mientras que las entradas *B* y *C* tienen 2 capas.

8.4**MEMORIA**

Es un espacio con la capacidad de almacenamiento de datos, en electrónica se relaciona por ejemplo con los Flip–flops.

8.5**RETARDOS**

Es la cantidad de tiempo que toma en viajar una señal desde el emisor hasta el receptor del circuito, la longitud de tiempo inicia cuando la entrada en una compuerta lógica se vuelve estable y a la vez, es estable a la salida de dicho circuito. Al reducir los retardos en un circuito digital se permite al procesar los dados a una mayor velocidad y mejorar el rendimiento en general.

CAPÍTULO 9. COMPLEJIDAD CIRCUITAL

Sea *B* (Base completa) un conjunto de funciones booleanas y *f* una función booleana, la cual es definida como $C_B(f)$. La cantidad de $C_B(f)$, se denomina complejidad circuital de *f* sobre *B.* para los circuitos con un Fan-in limitado, existen dos métodos generales; la primera es una base denominada B_2, son las funciones booleanas las cuales son de 2bits. Esto hace que la complejidad circuital sea un problema muy natural y la pregunta en general es ¿Qué tan eficiente se puede construir $f \in B_n$ para funciones B_2? Otra base es U_2(Base estándar), la cual habla informalmente de las compuertas lógicas, U_2 consiste en 8 diferentes funciones; cuando se tienen contantes como 0 y 1 en el circuito. Al conectar constantes en las funciones de U_2 se puede obtener $f(x,y) = x, f(x,y) = \bar{x}, f(x,y) = y, f(x,y) = \bar{y}, f(x,y) = 0$ y $f(x,y) = 1$. Las cuales son seis funciones, más lo que da un total de 14. La ventaja de U_2 conrespecto B_2 es que a B_2 le faltan dos funciones $XOR(x,y) = (x \oplus y)$ y $XNOR(x,y) = \overline{x \oplus y}$.

9.1 MODELOS Y MEDIDAS DE CIRCUITOS

En esta sección se va a hablar de las medias de complejidad, las relaciones entre las medidas y cómo afecta cualquier cambio, ya sea en la función o en el circuito y los diferentes métodos para obtener el límite inferior.

9.1.1 Modelos de circuito

Los circuitos lógicos son circuitos programados en línea recta, ya que sus variables son 0 y 1. El fan-in de una base es el máximo fan-in de cualquier base en el circuito; la cual generalmente las compuertas lógicas se hace uso de un fan-in de 2 {AND, OR}. Un fan-in es de base completa cuando toda función binaria puede ser computada sobre U.

El fan-out de 1 juega un papel importante con respecto a la profundidad del circuito; Cada circuito de fan-out de 1 corresponde a una de las características requeridas por los vértices del circuito. En la **Figura 9.1.1.1** se observa un circuito con un fan-out de 1 y un fan-in de 2 y haciendo uso de la compuerta AND

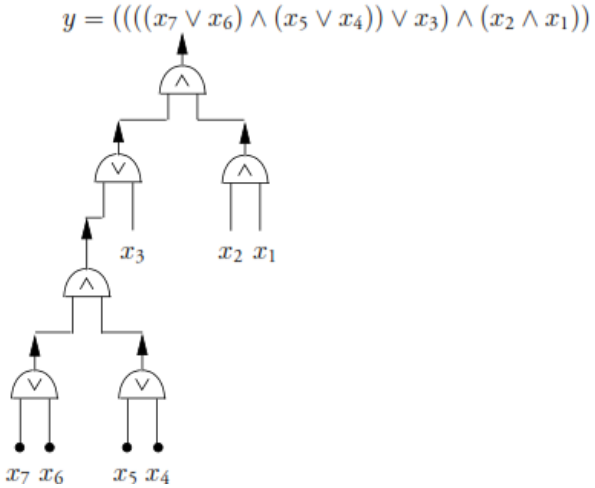


Figura 9.1.1.1 ejemplo de fan-in y fan-out

El circuito de profundidad limitada es un circuito sobre la base estándar U_2, donde el fan-in de las compuertas AND y OR puede ser ilimitado, pero la profundidad es limitada. También se hace referencia a circuitos monótonos sobre la base $U_{mon} = \{AND, OR\}$ en el cual el fan-in es de 2

9.1.2 **Medidas de complejidad**
Ahora se va a definir las medias de complejidad requeridas.

9.1.2.1 **Tamaño del circuito**
El tamaño del circuito de una función binaria $f: B^n \mapsto B^m$ con respecto a la base U, llamada $C_U(f)$, es de menor número de compuertas en cualquier circuito para f sobre la base U. El tamaño del circuito con un fan-out s, se llama $C_{s,U}(f)$, el cual es un circuito de tamaño de f cuando el fan-out del circuito es limitado como máximo a s.

9.1.2.2 **Profundidad del circuito**
La profundidad de una función binaria $f: B^n \mapsto B^m$ con respecto a la base Ues llamada $D_U(f)$, es de menor profundidad en un circuito para f sobre la base U. La profundidad del circuito con un fan-out s, es llamado $D_{s,U}(f)$, el cual es un circuito de profundidad de f cuando el fan-out del mismo es limitado como máximo a s.

9.1.2.3 **El tamaño de la fórmula**
El tamaño de la fórmula en una función booleana $f: B^n \mapsto B$ con la base respectiva U, es llamada $L_U(f)$, es el número mínimo de vértices de cualquier circuito con un fan-out de 1 de f sobre la base U.
Vale comentar que la diferencia entre la fórmula y el tamaño de un circuito, es que la fórmula cuenta el número de vértices de entrada, mientras que en el tamaño se cuenta con la cantidad de compuertas usadas.

9.2**RELACIONES ENTRE MEDIDAS**
Se analiza el efecto que tiene en la complejidad las medidas cuando se hace un cambio en la base o el fan-out de un circuito y se va a tocar a fondo la relación entre la profundidad y el tamaño de fórmula en un circuito.

9.2.1 **Efecto del fan-out en el tamaño del circuito**
En un circuito el tamaño y la profundidad de una función, cambia a medida que se va reduciendo el fan-out máximo del mismo. Esto es de gran relevancia para la medida de la complejidad y el uso de tecnologías, las cuales limitan el fan-out de las compuertas.
LEMA 1: Un árbol con un máximo fan-in r, con k vértices tiene como máximo $k(r-1)+1$ hojas; un árbol con l hojas y un fan-in r tiene como máximo $l-1$ vértices con fan-in de 2 o más y como máximo el árbol tiene $2(l-1)$ aristas.
LEMA 2: permitiendo que U sea una base del fan-in r. Por cada $f: B^n \mapsto B$ las siguientes desigualdades permanecen entre el tamaño de la fórmula $L_U(f)$ y el tamaño del circuito $C_{1,U}(f)$ y un fan-out de 1 se tiene que $\frac{L_U(f)-1}{r-1} \leq C_{1,U}(f) \leq 3L_U(f)-2$
La primera diferencia se diverge de la definición del tamaño de la fórmula y del primer resultado que se expresó en el **LEMA 1,** en donde $k = C_{1,U}(f)$. La segunda divergencia también se deduce del **LEMA 1**, donde un árbol con $L_U(f)$ hojas tiene como máximo $L_U(f)-1$ vértices con un fan-in de 2 o más y como máximo $2(L_U(f)-1)$ aristas entre vértices (en donde se incluyen las hojas). Cada una de estas aristas pueden llevar la compuerta NOT, al igual que una compuerta de salida, para un total como máximo de $2L_U(f)-1$ de compuertas NOT. De este modo un circuito con un fan-out de 1, tiene como máximo $3L_U(f)-2$ compuetas.

Como se muestra ahora, el tamaño del circuito aumenta como máximo un valor constante cuando el fan-out del circuito se reduce a s para $s \geq 2$. La mayoría de las compuertas son necesarias para calcular la función indefinida $i(x) = x$, explicando lo anterior: sí una base contiene las compuertas AND u OR, la función de identidad puede ser obtenida uniendo las dos entradas a la misma fuente.
Se termina sí U contiene una función tal, que al fijar todas las variables a excepción de una, se pueda calcular $i(x)$. Si no, se busca una función no monótona en U. Dado que algunos binarios no son monótonas (como lo es $\bar{x}$), una función g en una base completa U es no monótona. Esto quiere decir, que existen tuplas x y y para g, $x \leq y$ tales que $g(x) = 1 > g(x) = 0$. Sea u y v las tuplas más grandes y las más pequeñas respectivamente, satisfaciendo $x \leq u \leq v \leq y$ y $g(u) = 1$ y $g(v) = 0$. Por lo tanto u y v difieren de una posición como máximo. Explicando generalmente, se deja que esa posición sea la primera y permita que los valores en las posiciones restantes en ambas tuplas de $(c_2, \dots, c_n)$. De ello se deduce que $g(1, c_2, \dots, c_n) = 0$ y $g(0, c_2, \dots, c_n) = 1$ o $g(x, c_2, \dots, c_n) = \bar{x}$. Si $l(U)$ es el número de compuertas de U necesarias para realizar una función idéntica, entonces $l(U) = 1$ o 2.

Teorema 9.2.1 Sea U una base completa con fan-in r y sea $f: B^n \mapsto B^m$. Las siguientes desigualdades se sostienen en $C_{s,U}(f)$.

$$C_U(f) \leq C_{s+1,U}(f) \leq C_{s,U} \leq C_{1,U}(f)$$

Además, $C_{s,U}(f)$ tiene relación con $C_U(f)$ para $s > 2$;

$$C_{s,U}(f) \leq C_U(f)\left(1 + \frac{l(U)(r-1)}{s-1}\right)$$

El primer conjunto de desigualdades se mantiene porque el circuito pequeño con un fan-out s, no es más pequeño que otro pequeño con un fan-out de $s + 1$, pero es un tipo de circuito menos restrictivo.
Otra desigualdad se construye un árbol con funciones de identidad en cada compuerta cuyo fan-out excede a s como se muestra en la **Figura 9.2.1.1**:

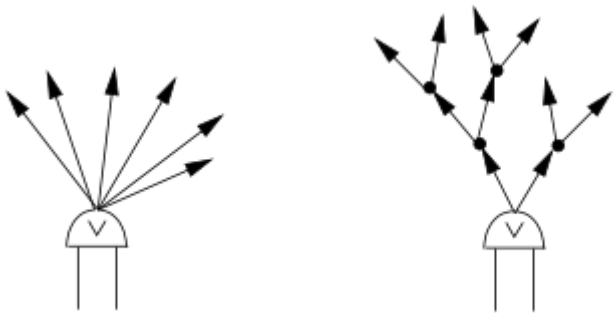


Figura 9.2.1.1 Evolución de un vértice con un fan-out más que s a un sub-arbol con un fan-out de s, donde $s = 2$.

Si una compuerta tiene un fan-out $\phi > s$, el fan-out se reduce a s y después conecta una compuerta de identidad a una de las s salidas. Esto aumenta el fan-out de s a $s + s - 1$. Si ϕ es mayor que este número, se repite el proceso de adición de una compuerta de identidad k veces, donde k es un entero más pequeño tal que $s + k(s-1) \geq \phi$ o es el entero más grande, tal que $s + (k-1)(s-1) < \phi$. Así $k < \frac{\phi - 1}{s-1}$.
Sea ϕ_i, el fan-out de la compuerta i en un circuito para f con un fan-out que es potencialmente ilimitado y sea k_i un entero más grande el cual satisfaga el siguiente límite:

$$k_i < \frac{\phi_i - 1}{s - 1}$$

Por lo tanto, la mayoría de las compuertas $\sum_i (k_i l(U) + 1)$ se necesitan un circuito con un fan-out de s para realizar a f, para la compuerta i en el circuito original y $k_i l(U)$ compuertas para las k_i, las cuales son las copias de la función de identidad en la i compuerta. Se tiene que tener en cuenta que $\sum_i \phi_i$ es el número de aristas dirigidas lejos de las compuertas del circuito original. Sin embargo, dado que cada arista va dirigido fuera de una compuerta, es una arista que va hacia otra compuerta, este número es como máximo $rC_U(f)$, ya que cada compuerta tiene un fan-in a lo sumo de r.

De lo dicho anteriormente se deduce que el número más pequeño de compuertas en un circuito con un fan-out s para f satisface el siguiente límite:

$$C_{s,U}(f) \leq C_U(f) + l(U) \sum_{i=1}^{C_U(f)} \left(\frac{\phi_i - 1}{s - 1}\right) \leq C_U(f)\left(1 + \frac{l(U)(r-1)}{s-1}\right)$$

Donde se demuestra que el tamaño del circuito con un fan-out de $s \geq 2$ se diferencia del tamaño del circuito con un ilimitado fan-out por lo menos, en un factor constante.

Con la construcción empleada en el teorema anterior, se puede decir, que el límite superior en $D_{s,U}(f)$, el cual es proporcional al producto de $D_U(f)$ y al $\log C_U(f)$. El límite superior indicado anteriormente sobre $C_{s,U}(f)$ se puede obtener mediante un circuito que también tenga un límite superior en $D_{s,U}(f)$ el cual es proporcional a $D_U(f)$ y a $\log_r s$.

9.2.2 Cambio de base y el efecto en el tamaño y profundidad del circuito.

LEMA 3: teniendo 2 bases completas U_a, U_b y una función $f: B^n \mapsto B^m$, el tamaño y la profundidad del circuito de f en estas dos bases difieren por los factores multiplicativos más constantes.

Explicación. Cada base es completa, toda función en U_a puede ser calculado por un número fijo de compuertas en U_b, y de forma viceversa. Dado un circuito con base U_a, un circuito con base U_b puede ser construido reemplazando cada compuerta de U_a por un número fijo de compuertas de U_b. Este tiene como efecto el aumento del tamaño del circuito, como máximo un factor constante. Se deduce que $C_{U_a}(f) = \Theta\left(C_{U_b}(f)\right)$. Dado que esta construcción también aumenta la profundidad como máximo un factor constante, de deduce que $D_{U_a}(f) = \Theta\left(D_{U_b}(f)\right)$.

9.2.3 Tamaño de fórmula vs profundidad del circuito

Si la fórmula es representada por un árbol balanceado, esto significa que el fan-in del circuito está limitado. Sin embargo, no se puede garantizar que cada fórmula corresponde a un árbol equilibrado, al suceder esto, se busca una forma de equilibrarlo.

Para equilibrar la fórmula y proporcionar un límite de profundidad del circuito de una función en términos del tamaño de la fórmula, se hace uso de la función de un multiplexor $f_{mux}^{(n)}: B^{2^n+n} \mapsto B$ en tres entradas $f_{mux}^{(1)}(a, y_1, y_0)$. Ahí el valor de a determina cuál de los dos valores se devuelve.

$$f_{mux}^{(1)}(a, y_1, y_0) = \begin{cases} y_0 & a = 0 \\ y_1 & a = 1 \end{cases}$$

Esta función puede ser realizada por

$$f_{mux}^{(1)}(a, y_1, y_0) = (\bar{a} \wedge y_0) \vee (a \wedge y_1)$$

Teniendo una base U con un fan-in r, la constante $d(U)$ se define como:

$$d(U) = \left(D_U\left(f_{mux}^{(1)}\right) + 1\right) / \log_r\left(\frac{r+1}{r}\right)$$

La medida $d(U)$ de una base U, es utilizada para obtener los límites en la profundidad del circuito de una función en términos del tamaño de la fórmula.

El teorema del separados para árboles, indica que un árbol puede ser dividido en dos árboles de aproximadamente el mismo tamaño mediante la eliminación de una arista. En el **LEMA 4** se establece una propiedad del teorema para la separación de árboles.

LEMA 4: Sea T un árbol con n vértices internos. Si el fan-in de cada vértice de T es como máximo r, entonces para k en donde $1 \leq k \leq n$, donde T tiene un vértice v tal que sub-arbol T_v enraizado en v que tiene k hojas pero cada uno de sus hijos $T_{v1}, T_{v2}, \ldots, T_{vp}$ para $p \leq r$, tiene al menos k hojas.

Explicación: Si la propiedad de la raíz se sostiene, el resultado es el mismo, pero sí no es así uno de los sub-árboles de T , que tengan por lo menos k hojas y se hace la prueba recursivamente. Debido a que un vértice de hoja tiene un vértice de hoja del sub-árbol del mismo, esto termina en algún vértice v el cual mantiene la propiedad. Si termina en un vértice de hoja, cada uno de los hijos es un árbol vacío.

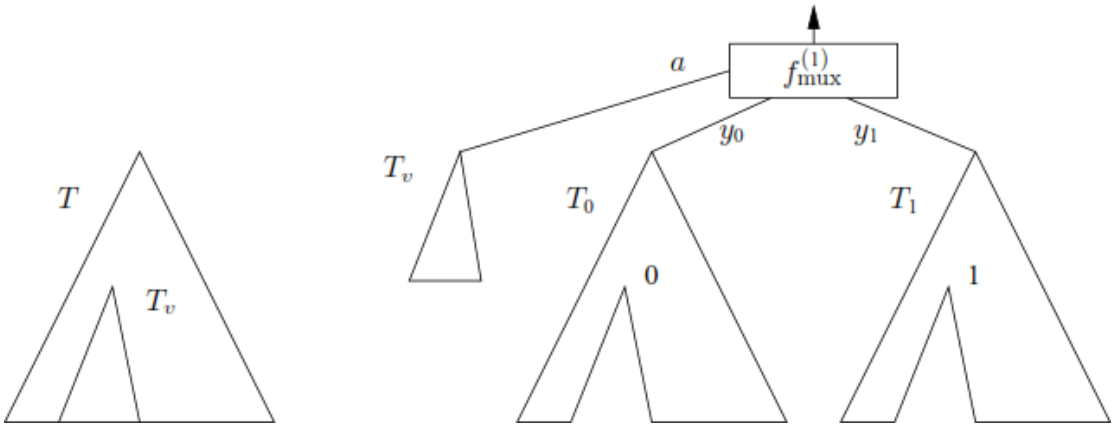


Figura 9.2.3.1 la descomposición del árbol T con el propósito de reducir la profundidad.

En la **Figura 9.2.3.1** se observa con se elimina el árbol grande T y se utiliza su valor para seleccionar el valor calculado por dos arboles formados a partir del árbol original, en el cual se sustituye el valor de T por 1 y 0 alternativamente.

9.3 MÉTODO DEL LÍMITE INFERIOR EN CIRCUITOS GENERALES

Se presentan diferentes métodos para derivar el límite inferior en el tamaño y la profundidad del circuito para funciones particulares las cuales son desarrolladas por circuitos lógicos generales.

9.3.1 Limites inferiores sencillos

Una función $f: B^n \mapsto B$ con n variables depende de la i variable x_i, sí existen valores $c_1, c_2, \ldots, c_{i-1}, c_{i-2}, \ldots, c_n$ tales que:

$$f(c_1, c_2, \ldots, c_{i-1}, 0\,, c_{i-2}, \ldots, c_n) \neq f(c_1, c_2, \ldots, c_{i-1}, 1\,, c_{i-2}, \ldots, c_n)$$

Esta propiedad lleva a límites inferiores el tamaño y profundidad del circuito que se obtiene al ser conectado, el cual, debe tener para calcular la función dependiendo de cada una de las variables.

Este límite inferior es el mejor posible teniendo en cuenta la información que se usa para derivar la misma. Para poder verlo, se debe observar la función $f(x_1, x_2, \dots, x_n) = x_1 \wedge x_2 \wedge \dots \wedge x_n$, el cual depende de cada una de sus variables, el circuito tiene de tamaño $\lceil (n-1) / (r-1) \rceil$ y como profundidad $\lceil \log_r n \rceil$ sobre la base la entrada r que tiene la compuerta AND.

9.3.2 **Método de eliminación de compuerta para el tamaño del circuito**

En este método se es necesario que las funciones sean lineales. Si se aplica con atención, él proporciona los límites inferiores más fuertes conocidos para bases completas. Este método utiliza el estímulo de las propiedades de una función f de n variables para mostrar dos cosas: primero es que se pueda asignar algunas variables de f para que la función resultante sea del mismo tipo que f y la segunda es que un par de compuertas en cualquier circuito para f puede ser eliminado mediante la asignación de valores. Después de eliminar todas las variables asignándoles valores, la función es constante. Dado que el número de compuertas del circuito original no puede ser menor que el número de compuertas eliminadas durante el proceso, el circuito original tiene por lo menos el mismo número de compuertas que se han retirado.
Este método ha sido un fracaso dado que la mayoría de las funciones booleanas son de forma exponencial y como se vio desde el inicio uno de los requerimiento para aplicar este método es que la función debe ser de forma lineal.

9.4**MÉTODO DEL LÍMITE INFERIOR PARA EL TAMAÑO DE FÓRMULA**

Como las fórmulas corresponden a los circuitos con un fan-in de 1, el tamaño de fórmula de una función puede ser mucho mayor al tamaño del circuito. Una de sus propiedades es que cada uno de los límites es cuadrático o casi cuadrático con respecto al número de entradas.
Para definir la idea sobre el tamaño de la fórmula se construye un circuito con fan-out de 1 para la función de acceso de almacenamiento indirecto$f_{ISA}^{(k,l)}: B^{k+lk+L} \mapsto B$, donde $K = 2^k$ y $L = 2^l$:

$$f_{ISA}^{(k,l)}(\boldsymbol{a}, \boldsymbol{x}_{K-1}, \dots, \boldsymbol{x}_0, \boldsymbol{y}) = y_{|\boldsymbol{x}_{|\boldsymbol{a}|}|}$$

Aquí $\boldsymbol{a}$ es una tupla de k, $\boldsymbol{x}_j = (x_{j,l-1}, \dots, x_{j,0})$ es una tupla de l para $0 \leq j \leq K-1$ y $\boldsymbol{y} = (y_{L-1}, \dots, y_0)$ es una tupla de L. El valor de $f_{ISA}^{(k,l)}$ es calculado indirectamente; es decir, el valor de $\boldsymbol{a}$ es interpretado como un número binario con un valor $|\boldsymbol{a}|$ el cual es usado para seleccionar el $|\boldsymbol{a}|$ de la tupla l de $\boldsymbol{x}_{|a|}$; esto, a su vez, es tratado como un número binario y su valor es usado para seleccionar la $|\boldsymbol{x}_{|a|}|$th variable en $\boldsymbol{y}$.

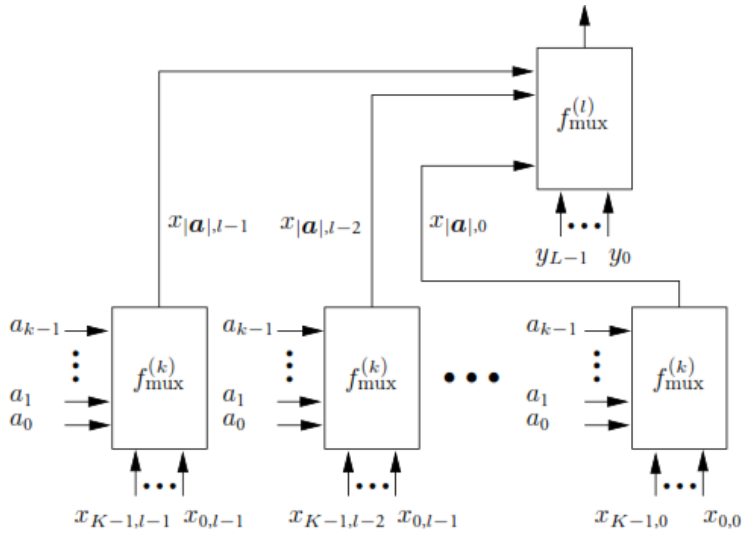


Figura 9.4.1 Esquema usado para la construcción de un circuito con fan-out de 1 para el almacenamiento de acceso indirecto en la función de $f_{ISA}^{(k,l)}$

9.4.1 El límite inferior de Krapchenko

Este método se aplica para una base común U_0 o cualquier sub-conjunto completo, es decir $\{\wedge, \neg\}$ y $\{\vee, \neg\}$.

Se hace uso del método de Krapchenko en una función de paridad $f_{\oplus}^{(n)}: B^n \mapsto B$, donde $f_{\oplus}^{(n)}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$, para mostrar que el tamaño de la fórmula es cuadrático en n. Dado que la función de paridad en dos variables pueden ser expresadas mediante la fórmula

$$f_{\oplus}^{(2)}(x_1, x_2) = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2)$$

Dado dos conjuntos disjuntos $A, B \subseteq \{0,1\}^n$ del conjunto de tuplas n, el vecindario (*neighborhood*) de A y B, $\mathcal{N}(A, B)$, es un conjunto pares de tuplas $(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x} \in A$ y $\boldsymbol{y} \in \boldsymbol{B}$, tal que $\boldsymbol{x}$ y $\boldsymbol{y}$ coincida en todas la posiciones exceptuando a una. El vecindario de $A = \{1\}$ y $B = \{0\}$ es par de $\mathcal{N}(A, B) = \{(1,0)\}$. También el vecindario de $A = \{001, 111\}$ y $B = \{110, 000\}$lo cual es par $\mathcal{N}(A, B) = \{(001, 111), (110,000)\}$. Dada la función $f: B^n \mapsto B$, se usa la notación $f^{-1}(0)$ y $f^{-1}(1)$ para denotar conjuntos de tuplas de n lo cual hace que f asuma los valores de 0 y 1 respectivamente

9.5 MÉTODO DEL LÍMITE INFERIOR PARA CIRCUITOS MONÓTONOS

Los mejores límites inferiores que se obtienen son los que se derivan en el tamaño del circuito sobre las bases completas en bases de funciones booleanas en n variables las cuales son lineales en n. Así mismo, el mejor límite inferior en el tamaño de fórmula que se han obtenido sobre bases completas son el mejor de los casos cuadráticos en n. Debido a esto, en la excursión de encontrar los mejores límites inferiores se ha llegado a estudiar los circuitos monótonos, donde la base es U_{mon}, para funciones monótonas y el resultado de esto ha sido que algunas de las funciones monótonas tienen un tamaño de circuito exponencial. Dado que la mayoría de las funciones booleanas en n variables tienen un tamaño de circuito $\Theta\left(2^n/n^{3/2}\right)$, para obtener el límite inferior en circuitos monótonos se tocaran de 3 métodos.

9.5.1 **Método de eliminación de trayectoria**

En este método se comprueba que se puede eliminar una trayectoria de compuertas sólo con fijar una variable de entrada en circuitos monótonos. Se le aplica el método a dos problemas uno es para la clasificación binaria y el otro es combinación binaria.

Problema 1: la clasificación binaria son funciones simétricas la cual la función es estándar en un circuito de tamaño lineal en n. Teniendo lo anterior en cuenta para calcular una función de clasificación binaria $f_{sort}^{(n)}: B^n \mapsto B^n$ que reorganiza los bits de una cadena binaria de n entradas de forma descendiente. De esa forma, la salida es 1 si una o más estradas es 1. Debido a esto se expresa de la forma $f_{sort}^{(n)}(x_1, x_2, \dots, x_n) = \left(\tau_1^{(n)}, \tau_2^{(n)}, \dots, \tau_n^{(n)}\right)$, donde $\tau_t^{(n)}$ es la función límite en n entradas con un límite t cuyo valor es 1, sí t o más de sus entradas es 1 y sería 0 en el caso contrario.

Teorema 9.5.1 el tamaño de un circuito monótono para $f_{sort}^{(n)}$ satisface los siguientes límites

$$n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} \leq C_{U_{mon}}\left(f_{sort}^{(n)}\right) = O(n \log n)$$

Para derivar cualquier límite inferior, se muestra que cualquier circuito para $f_{sort}^{(n)}$ donde hay una variable de entrada la cual se pueda establecer en 1, permitiendo que al menos $\lceil \log_2 n \rceil$ compuertas a lo largo de una de las trayectorias a la salida $\tau_1^{(n)}$ para ser removida del circuito y convertir el circuito en $f_{sort}^{(n-1)}$, como resultado, se muestra la siguiente expresión:

$$C_{U_{mon}}\left(f_{sort}^{(n)}\right) \geq C_{U_{mon}}\left(f_{sort}^{(n-1)}\right) + \lceil \log_2 n \rceil$$

Dejando $x_j = 0$ por $j \neq i$ pero dejando que x_i varíe. Las únicas funciones calculadas por las compuertas son 0, 1 o x_i; los valores de $\tau_1(\boldsymbol{x})$ en dichas entradas son iguales a x_i. Debido a esto, existe una trayectoria P desde el vértice marcado x_i a τ_1 tal, que en cada compuerta en la trayectoria se calcula la función x_i. Como se muestra en la **Figura 9.5.1.1.1**. Así, si $x_i = 1$ cuando $x_j = 0$ para $j \neq i$ la salida de cada una de las compuertas será 1, el valor de τ_1 y de cada uno de las compuertas de la trayectoria P de x_i permanece en 1 y puede ser removida. Al hacer este cambio de x_i también se puede reducir el límite de todas las funciones de salida por 1 y esto implica que el circuito ya calcula la función de clasificación binara con una variable menos. En la **Figura 9.5.1.1.1** se tiene cuando $x_i = 1$ hay una trayectoria P a τ_1 donde cada compuerta en P tiene como valor 1.

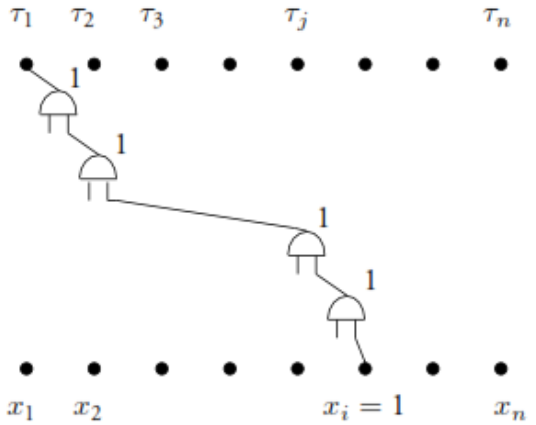


Figura 9.5.1.1 Ejemplo método de eliminación de trayectoria problema 1

Problema 2: para todo circuito de combinación binaria, tiene un tamaño, el cual es $U(n \log n)$. La combinación binaria es realizada por una función $f_{merge}^{(n)}: B^n \mapsto B^n$ y $n = 2k$, dados dos

tuplas k ordenadas $\boldsymbol{x}$ y $\boldsymbol{y}$, el valor de $f_{merge}^{(n)}$ es de n-tuplas, el cual resulta de ordenar n-tupla formado mediante la concatenación de $\boldsymbol{x}$ y $\boldsymbol{y}$. Basado en lo anterior, un circuito de combinación binaria se puede obtener a partir de uno para la clasificación, simplemente restringiendo los valores asumidos por las entradas del circuito de clasificación. La combinación binaria es una sub-función de la clasificación binaria.

9.5.1 Conservando a n igual. Entonces, el tamaño del circuito monótono para $f_{merge}^{(n)}: B^n \mapsto B^n$ satisface el siguiente límite:

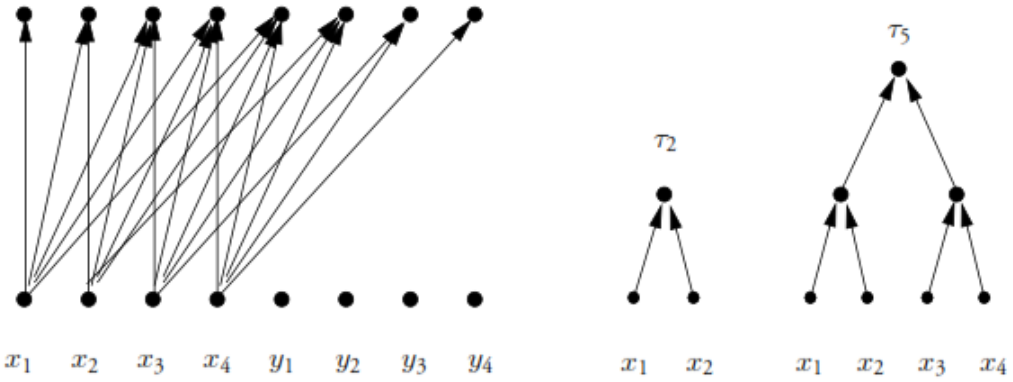
$$\left(\frac{n}{2}\right)\log_2 n - O(n) \leq C_{U_{mon}}\left(f_{merge}^{(n)}\right) = O(n \log n)$$

Conservando $k = n/2$ la función $f_{merge}^{(n)}$ trabaja con dos k-tuplas $\boldsymbol{x}$ y $\boldsymbol{y}$ para producir un resultado $f_{merge}^{(n)}(\boldsymbol{x}, \boldsymbol{y})$, donde $\boldsymbol{x}$ y $\boldsymbol{y}$ están en un orden descendiente,$x_1 \geq x_2 \geq x_3 \geq \cdots \geq x_k$ y $y_1 \geq y_2 \geq y_3 \geq \cdots \geq y_k$. Como se menciona anteriormente, para la clasificación binaria, las funciones de salida son $\tau_1, \tau_2, \tau_3, \ldots, \tau_n$.

Dejando $x_1 = x_2 = x_3 = \cdots = x_{r-1} = 1, x_{r+1} = \cdots = x_k = 0,\ y_1 = y_2 = y_3 = \cdots = y_s = 1,\ y_{s+1} = \cdots = y_k = 0$. En donde x_r no este especificado. Dado que el circuito es monótono, el valor calculado por cada compuerta del circuito es 0, 1 o x_r. También:

$$\tau_t(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & t < r + s \\ x_r & t = r + s \\ 0 & t > r + s \end{cases}$$

Se supone que debe existir una trayectoria $P_r^{(r+s)}$ de compuerta,s desde la entrada x_r a la salida τ_{r+s} tal, que la salida de compuerta es x_r. Así, los componentes de $\boldsymbol{x}$ se clasifican en $x_{r+1} = \cdots = x_k = 0$. Por el otro lado, si $x_r = 1$ dado la monotonicidad el valor de τ_{r+s} no puede ser cambiado por la variación de los valores $x_{r+1}, \ldots, x_k$. De este modo, τ_j es prácticamente dependiente de x_i para i y j que satisfacen $1 \leq i \leq k$ y $i \leq j \leq i + k$. Se sabe que existen trayectos los cuales son vértices-disjuntos entre x_1 y τ_{r+1}, x_2 y τ_{r+2},…, x_k y τ_{r+k} para $0 \leq s \leq k$ como se muestra en la



(a) (b)

Figura 9.5.1.2. Así, hay conjuntos de trayectos vértices-disjuntos que conectan las entradas $k = n/2$ en $\boldsymbol{x}$ y k en salidas consecutivas. En la

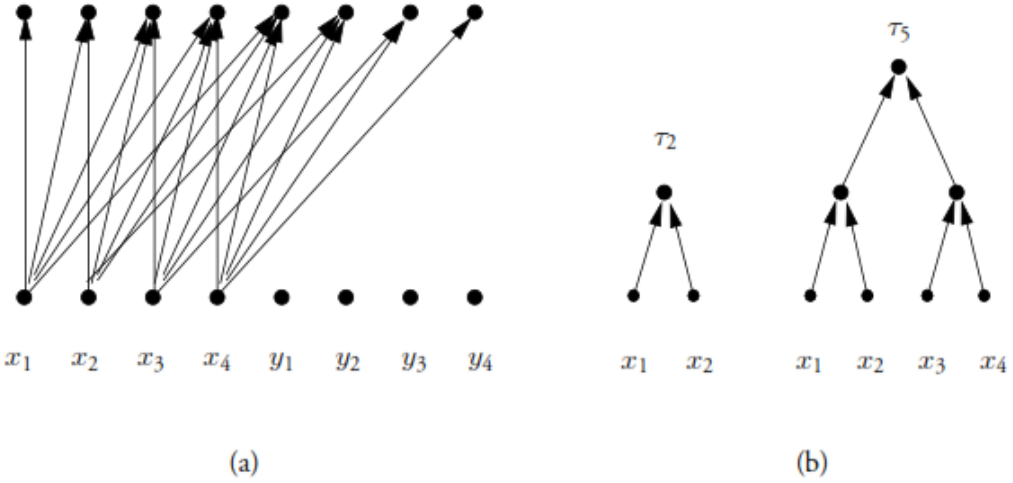


Figura 9.5.1.2 (a) es un circuito monótono para $f_{merge}^{(n)}$, $n = 2k$, $k + 1$ existen conjuntos de trayectos disjuntos k entre las entradas x y salidas k consecutivamente; los trayectos que conectan las entradas a cualquier salida forma un árbol binario. (b) los trayectos de la salida τ_j forman un árbol binario; el número de entradas de las cuales hay una trayectoria a τ_j se le llama $e(j)$, es el número de entradas de las cuales τ_j dependen

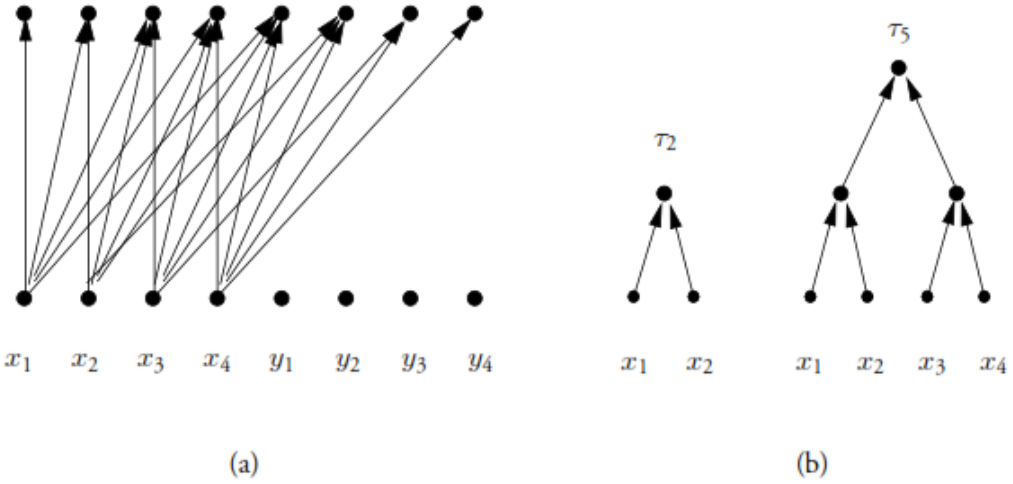


Figura 9.5.1.2 circuitos monótonos

Para derivar el límite inferior en $C_{U_{mon}}\left(f_{merge}^{(n)}\right)$, deja $d(i,j)$ indica la longitud la cual es el número de aristas o vértices sin entradas (compuertas); en el trayecto más corto desde la entrada x_i hasta la salida τ_j. Notoriamente, $d(i,j) = 0$ a menos que $i \leq j \leq i + k$. Dado que la trayectoria tiene una longitud tal menos tan grande como $d(i,j)$ se deduce que $C_{U_{mon}}\left(f_{merge}^{(n)}\right)$ satisface el siguiente límite:

$$C_{U_{mon}}\left(f_{merge}^{(n)}\right) \geq max\left\{\sum_{r=1}^{k} d(r, r+s) | 0 \leq s \leq k\right\}$$

Como el máximo de un conjunto de enteros es al menos igual al promedio de dichos enteros, se tiene que para $k = n/2 \geq 1$:

$$C_{U_{mon}}\left(f_{merge}^{(n)}\right) \geq \frac{1}{k+1}\sum_{s=0}^{k}\sum_{r=1}^{k} d(r, r+s) = \frac{1}{k+1}\sum_{j=1}^{2k}\sum_{i=1}^{k} d(i,j)$$

De la última identidad se continúa usando el hecho de que $d(i,j) = 0$ a menos que $i \leq j \leq i+k$. Pero $\sum_{i=1}^{k} d(i,j)$ es la suma de las distancias de las trayectorias más cortas de las entradas relevantes de $\boldsymbol{x}$ a la salida τ_j para $1 \leq j \leq 2k$. Dado a los trayectos de un árbol binario y que τ_j depende de las entradas $e(j)$, esta es la longitus de la trayectoria externa del árbol con $e(j)$ hojas. La trayectoria externa es al menos $e(j)\lceil \log_2 e(j) \rceil - 2^{\lceil \log_2 e(j) \rceil} + e(j)$. Debido a lo anterior el tamaño del circuito monótono más pequeño es el siguiente límite inferior cuando $n = 2k$:

$$C_{U_{mon}}\left(f_{merge}^{(n)}\right) \geq \frac{1}{k+1}\sum_{j=1}^{2k}[e(j)\log_2 e(j)]$$

$$= \frac{2}{k+1}\sum_{j=1}^{k}[j \log_2 j]$$

En la última expresión se hace uso de la definición de $e(j)$.

9.5.2 **Método de reemplazo de funciones**

Este método simplifica los circuitos monótonos reemplazando una función calculada en el vértice interno, por una nueva función, sin cambiar la función calculada por el circuito general. Dado que en un paso de reemplazo se eliminan compuertas y se reduce el problema a un sub-problema, el método proporciona bases para establecer el límite inferior en la complejidad del circuito haciendo uso de la prueba por inducción.

Se describe el método de reemplazo y luego se aplican en una convolución booleana y una multiplicación de matriz booleana.

Una **regla de reemplazo** es una regla, la cual permite que una función calculada en un vértice por otra sin cambiar la función calculada por el circuito. Antes de establecer cualquier regla se debe tener en cuenta:

Definicion 9.5.1 [6] Sea $\boldsymbol{x}$ las variables de una función booleana $f: B^n \mapsto B$ un implicante de f es un producto (AND) π, de un sub-conjunto de los lineales de f (sus variables y complementos) tal que sí $\pi(\boldsymbol{x}) = 1$ en la entrada n-tupla $\boldsymbol{x}$, entonces $f(\boldsymbol{x}) = 1$ para $\pi \leq f$. El conjunto de implicantes de una función f se denota $I(f)$.

Un implicante π de una función booleana f es un implicante primo si no hay un implícito π_1 diferente de π tal que $\pi \leq \pi_1 \leq f$. El conjunto de implicantes primos de una función f se denota $PI(f)$.

Un implicante monótono (monómero) de una función booleana monótona $f: B^n \mapsto B$ es un producto (AND) π de las variables incompletas de f tales que si $\pi(\boldsymbol{x}) = 1$ en la entrada n-tupla $\boldsymbol{x}$ entonces $f(\boldsymbol{x}) = 1$. El monómero vacío tiene como valor 1. El conjunto de implicantes de una función f es nombrada $I_{mon}(f)$.

Un implicante monótono π de una funcion booleana f es un implicante primo monótono si no hay un implicante monótono π_1 diferente de π tal que $\pi \leq \pi_1 \leq f$. El conjunto de implicantes primos monótonos de una función f es denotado $PI_{mon}(f)$.

Los productos de la expansión de la suma de productos (SOPE) son implicantes no-monótonos de una función booleana. Los implicantes primos de una función booleana la definen completamente. En caso de una función booleana monótona, los implicantes primos son implicantes primarios monótonos.

La función $c_{j+1} = (p_j \wedge c_j) \vee g_j$ usada en el diseño de un Full-adder es una funcion monótona con variables p_j, c_j y g_j. Su conjunto de implicantes es $c_{j+1} = \{p_j \wedge c_j\ , g_j, p_j \wedge g_j\ , c_j \wedge g_j,$ $p_j \wedge c_j \wedge g_j\}$. Sí cualquiera de estos productos tiene un valor de 1 entonces también lo hace c_{j+1}. Sí el conjunto de implicantes primos es $PI(c_{j+1}) = \{p_j \wedge c_j, g_j\} \subseteq I(c_{j+1})$ porque estos son los productos más pequeños para los cuales c_{j+1} tiene un valor de 1. De esta forma c_{j+1} es definido por $PI(c_{j+1})$ y es representado por $c_{j+1} = (p_j \wedge c_j) \vee g_j$.

Convolución booleana la función de convolución booleana $f_{conv}^{(n)}: B^{2n} \mapsto B^{2n-1}$ mapea n-tuplas $\boldsymbol{a} = (a_0, a_1, \dots, a_{n-1})$ y $\boldsymbol{b} = (b_0, b_1, \dots, b_{n-1})$ sobre $(2n-1)$-tupla $\boldsymbol{c}$, denotado $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$, donde $c_j \;\; 0 \leq j \leq 2n-2$ y es definida:

$$c_j = \sum_{r+s=j} a_r \wedge b_s$$

La convolución booleana puede ser realizada por un circuito sobre una base estándar U_o para multiplicar números binarios de la siguiente manera. $\boldsymbol{a}$ y $\boldsymbol{b}$ representados por los siguientes números enteros donde $q = \lceil \log_2 n \rceil + 1$:

$$a = \sum_{k=0}^{n-1} a_i 2^{qi}\,, \qquad\qquad b = \sum_{j=0}^{n-1} b_j 2^{qj}$$

Lo que quiere decir es que cada bit de a y b esta separado por $\lceil \log_n n \rceil$ ceros.

$$ab = \sum_{k=0}^{2n-2} \left(\sum_{i+j=k} a_i b_j \right) 2^{qk}$$

Debido a que ninguna suma interna en la expresión anterior es mayor a $2n-1$, y q bits es suficiente para representarla en notación binaria. Lógicamente, no hay acarreo entre las dos sumas internas. Se deduce que cualquier suma interna no cero sí y solo sí $c_k = 1$. Precisamente, el valor de c_k puede ser obtenido formando el OR de los bits en las posiciones $k_q, k_q + 1, \dots, k_q + q - 1$ del producto. Dado a que dos tuplas binarias m pueden ser multipicadas en una notación binaria por un circuito de tamaño $O(m(\log m)(\log\log m))$, la función $f_{conv}^{(n)}$ puede ser calculad por el tamaño del circuito $O(n(\log^2 n)(\log\log m))$ en donde $m = nq = O(n \log n)$.

El objetivo es utilizar el método de sustitución de funciones para mostrar que cada circuito monótono para la convolución booleana tiene un tamaño $U(n^{3/2})$. Como se mencionó anteriormente. Cada paso de reemplazo elimina a implicantes primos de la función g calculada en alguna compuerta y cambia la función f calculada por el circuito. Si la nueva función f^* está en la misma familia que f, el reemplazo de la compuerta puede continuar y la inducción puede ser aplicada. Dado que la función de convolución no cambia necesariamente a otra instancia de sí misma y mucho menos sus variables, se coloca esta función en la clase de formas bilineales semi-disjuntas.

LEMA 5 ninguna compuerta de un circuito monótono de tamaño mínimo para una forma bilineal semi-disjunta de $f^{(n,m,p)}$ calcula una función g cuyos implicantes primos incluyen dos variables de $\boldsymbol{x}$ o $\boldsymbol{y}$. Una simple demostración por inducción en el tamaño del circuito demuestra que sí un circuito para $f^{(n,m,p)} = (f_1, f_2, \dots, f_p)$ contienen un cálculo de compuerta g entonces $f_r \;\; 1 \leq r \leq p$, puede ser escrito de la forma

$$f_r(\boldsymbol{x}, \boldsymbol{y}) = \big(p_r(\boldsymbol{x}, \boldsymbol{y}) \wedge g(\boldsymbol{x}, \boldsymbol{y})\big) \vee q_r(\boldsymbol{x}, \boldsymbol{y}) \blacksquare$$

La función de convolución booleana es de una forma semi-disjunta bilineal. Cada implicante de cada componente de $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ contiene una variable $\boldsymbol{a}$ y otra $\boldsymbol{b}$. Además los implicantes primos de c_i y c_j son disjuntos si $i \neq j$.
Ya que cada una de las n variables de entrada en $\boldsymbol{a}$ hay n funciones de salida en $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ que dependen de $d_i = n$ para $1 \leq i \leq n$ en donde d_i es el número de funciones en $\{f_1, f_2, \dots, f_p\}$ que en esencia depende de la variable de entrada $x_i,\ 1 \leq i \leq n$.

9.5.1 Sea $f_{conv}^{(n)}: B^{2n} \mapsto B^{2n-1}$ una función de convolición booleana. Entonces el tamaño de circuito monótono de $f_{conv}^{(n)}$ satisface el límite inferior:

$$C_{U_{mon}}\left(f_{conv}^{(n)}\right) \geq n^{3/2}$$

No se conoce ningún límite superior en el tamaño del circuito monótono de $f_{conv}^{(n)}$ que coincida con el límite inferior.

Multiplicación de matriz booleana una matriz $I \times J$ $A = [a_{i,j}]$, donde $1 \leq i \leq I$ y $1 \leq j \leq J$, se toman las entradas en una matriz para que sean variables booleanas.

9.5.2 Sea $A = [a_{i,k}]$ donde $1 \leq i \leq n$ y $1 \leq k \leq m$, $B = [b_{k,j}]$ dnde $1 \leq k \leq m$ y $1 \leq j \leq p$, y $C = [c_{i,j}]$ donde $1 \leq i \leq n$ y $1 \leq j \leq p$, sea las matrices $n \times m$, $m \times p$ y $n \times p$ respectivamente, el producto $C = A \times B$ donde A y B son funciones de $f_{MM}^{(n,m,p)}: B^{nm+mp} \mapsto B^{np}$ cuyos valores en la matrices A y B es la matriz C, cuya entrada en la fila i y la columna $j, c_{i,j}$ está definida como:

$$c_{i,j} = \bigvee_{k=1}^{m} a_{i,k} \wedge b_{k,j}$$

En donde AND ($\wedge$) y OR ($\vee$) son reemplazados por operadores de multiplicación y adición. La función de multiplicación matricial es de forma bilineal. Se asocia las entradas A con la tupla $\boldsymbol{x}$ y a B con la tupla $\boldsymbol{y}$. Para obtener el límite inferior en el número de OR necesarios para realizar un circuito monótono.

LEMA 6 todo circuito monótono para una matriz booleana $f_{MM}^{(n,m,p)}$ requiere al menos $n(m-1)p$ compuertas OR.

LEMA 7 todo circuito monótono para una matriz booleana $f_{MM}^{(n,m,p)}$ requiere al menos de nmp compuertas AND.

El algoritmo estándar para la función $f_{MM}^{(n,m,p)}: B^{nm+mp} \mapsto B^{np}$, la función de multiplicación de la matriz booleana es óptima. Utiliza nmp AND y $n(m-1)p$ OR.

9.5.3 **Método de aproximación**

Este método se usa para obtener grandes límites inferiores en el tamaño del circuito monótono para ciertas funciones booleanas monótonas. Este método convierte un circuito monótono $\hat{C}$ que calcula la función $\hat{f}$. Esto se hace reemplazando repetidamente una compuerta previamente no visitada más alejada de la compuerta de salida por una compuerta de aproximación la cual calcula una aproximación a la AND u OR que reemplaza. Cada operación de reemplazo cambia el circuito y aumenta una pequeña cantidad de tuplas de entrada en la cuales f y la función calculada por el nuevo circuito difieren. Cuando todo el proceso de reemplazo está completo, el circuito resultante se aproxima pobremente a f, es decir $\hat{f}$ y f se diferencian por una cantidad considerable de entradas. Para que esto suceda, el circuito monótono original bebe tener muchas

compuertas, cada una de ellas contribuyen con un número relativamente pequeño de errores al proceso de reemplazo completo. Realmente esta es la esencia del método de aproximación.

$\widehat{\wedge}$: La aproximación $f_l \,\widehat{\wedge}\, f_r$ a $f_l \wedge f_r$ se obtiene representando $f_l \wedge f_r$ haciendo uso de SOP y eliminando todos los términos del producto cuyo producto de puntos finales contienen más de p vértices. De esto se tiene que $f_l \wedge f_r \geq f_l \,\widehat{\wedge}\, f_r$.

$\widehat{\vee}$: La aproximación $f_l \,\widehat{\vee}\, f_r$ a $f_l \vee f_r$ se obtiene representando $f_l \vee f_r$ haciendo uso de POS y eliminando todos los términos del producto cuyo producto de puntos finales contienen más de q vértices. De esto se tiene que $f_l \vee f_r \leq f_l \,\widehat{\vee}\, f_r$.

PARTE III

Metodologías para el cálculo del límite inferior

CAPÍTULO 10. METODOLOGÍA DE REEMPLAZO DE FUNCÍON

1. Teniendo la función diseñar el circuito a base de compuertas lógicas básicas (AND, OR, NOT).
2. Obtener las medidas necesarias para calcular las medidas de complejidad como son: el tamaño, la profundidad y el tamaño de formula del circuito; también son necesarios calcular en fan-in y fan-out de las compuertas.
3. Aplicar el método de reemplazo de función en la función principal la cual se hace de la siguiente manera: reemplazar una sub-función de la ecuación principal por una constante *K*. teniendo en cuenta en hacer el reemplazo en donde todas sus variables sean positivas.
4. Calcular las nuevas medidas de complejidad y observar las diferentes variaciones que tuvo el circuito.

CAPÍTULO 11. METODOLOGÍA DE ELIMINACIÓN DE TRAYECTORIA

1. Teniendo la función principal, diseñar el circuito a base de compuertas lógicas básicas.
2. Obtener las medidas necesarias para calcular las medidas de complejidad.
3. Cambiar una de las variables de entrada por una constante (0 o 1).
4. Calcular las nuevas medidas de complejidad y observar las variaciones que tuvo el circuito.

PARTE IV

Aplicación

CAPÍTULO 12. APLICACIÓN DE LAS METODOLOGÍAS EN CIRCUITOS ARITMÉTICOS

Los circuitos aritméticos son aquellos que tienen como objetivo realizar operaciones aritméticas de forma binaria; desde el punto de vista para el procesamiento de datos las cuales pueden ser del tipo serie o paralela. Cuando se menciona de forma de serie se habla de que los datos se van presentando al circuito bit a bit comenzando por el LSB (bit menos significativo). De forma paralela, todos los bits se presentan de forma simultánea. Debido a lo anterior se va a tomar en cuenta un Full-adder como referente para un circuito aritmético serie, un sumador paralelo y la multiplicación de números binarios.

12.1 **FULL-ADDER**

Este circuito es una suma, cuenta con tres variables A y B, las cales son las entradas y C_i que es el acarreo; su tabla de verdad **Tabla 12-1-1**

A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabla 12-1-1 Tabla de verdad Full-adder

C_i \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Tabla 12-2 Mapa de Karnaugh S

C_i \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Tabla 12-3 Mapa de Karnaugh para C_o

$S = C_i \oplus (A \oplus B) = \bar{A}\bar{B}C_i + \bar{A}B\bar{C_i} + A\bar{B}\bar{C_i} + ABC_i;\ C_o = AB + BC_i + AC_i$

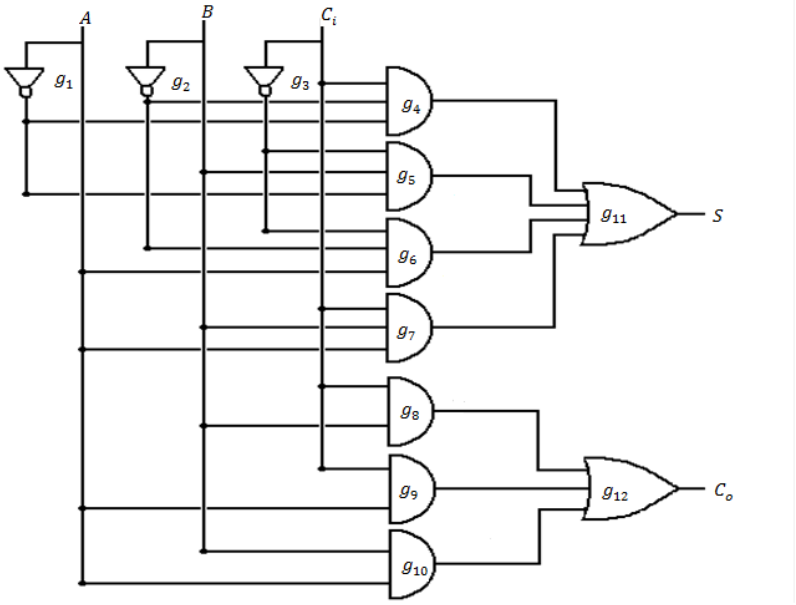


Figura 12.1.1 Circuito del Full-adder en compuertas AND, OR, NOT

$t_S = 3t_{gate},\ \ t_{C_o} = 2t_{gate}$

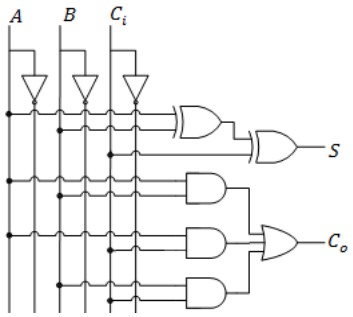


Figura 12.1.2 Circuito Full-adder minimizado

Teniendo la **Figura 12.1.1** y **Figura 12.1.2**, se puede observar que son el mismo circuito, pero la segunda figura está minimizando las compuertas en la función S haciendo que el tamaño del circuito y de fórmula se reduzca al mínimo pero no t_s dado que para cualquiera de los dos sigue siento $3t_{gate}$. Pero para obtener el límite inferior es necesario hace uso de la **Figura 12.1.1**.
Aplicando las medidas necesarias para medir la complejidad del circuito se tienen las siguientes:

- **Tamaño del circuito:** $C_{U_S}(f) = 8$
- **Tamaño del circuito** $C_{U_{Co}}(f) = 4$
- **Profundidad del circuito:** $D_{U_S}(f) = 3$
- **Profundidad del circuito:** $D_{U_{Co}}(f) = 2$
- **Tamaño de fórmula:** $L_{U_S}(f) = 7$
- **Tamaño de fórmula:** $L_{U_{Co}}(f) = 3$
- **Fan-out:** $A = 4,\ B = 4,\ C_i = 4,\ g_1 = 2,\ g_2 = 2,\ g_3 = 2,\ g_4 = 1,\ g_5 = 1,\ g_6 = 1,\ g_7 = 1,\ g_8 = 1,\ g_9 = 1,\ g_{10} = 1,\ g_{11} = 1,\ g_{12} = 1$
- **Fan-in:** $g_1 = 1,\ g_2 = 1,\ g_3 = 1,\ g_4 = 3,\ g_5 = 3,\ g_6 = 3,\ g_7 = 3,\ g_8 = 2,\ g_9 = 2,\ g_{10} = 2,\ g_{11} = 4,\ g_{12} = 3$

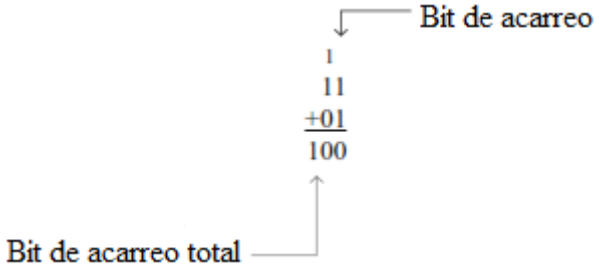
Para obtener el límite inferior del circuito se aplica el método de eliminación de compuertas en donde una sub-función $ABC_i = g_7 = K$ de S, se vuelve una constante; solo se toman sub-funciones donde todas sus variables son positivas, por ende esta compuerta es eliminada y algunas de las medidas de complejidad cambian y como se ve a continuación:

- **Tamaño del circuito:** $C_{U_S}(f) = 7$
- **Profundidad del circuito:** $D_{U_S}(f) = 3$
- **Tamaño de fórmula:** $L_{U_S}(f) = 6$
- **Fan-out:** $A = 3,\ B = 3,\ C_i = 3,\ g_1 = 2,\ g_2 = 2,\ g_3 = 2,\ g_4 = 1,\ g_5 = 1,\ g_6 = 1,\ K,\ g_8 = 1,\ g_9 = 1,\ g_{10} = 1,\ g_{11} = 1,\ g_{12} = 1$
- **Fan-in:** $g_1 = 1,\ g_2 = 1,\ g_3 = 1,\ g_4 = 3,\ g_5 = 3,\ g_6 = 3,\ K,\ g_8 = 2,\ g_9 = 2,\ g_{10} = 2,\ g_{11} = 3,\ g_{12} = 3$

Como se puede ver al aplicar este método se ve un cambio en el tamaño y el tamaño de fórmula del circuito, dado que son dependientes de las compuertas usadas. La profundidad no se ve afectado dado que esta medida se toma por el tramo más largo en el circuito es cuál es el que tiene más capas.

12.2 SUMADOR SEMI-PARALELO DE 4BITS

Como se evidenció anteriormente, un Full-adder es capaz de sumar dos números binarios de 1 bit y con un acarreo de entrada. Para sumar números binarios de más de 1 bit, se hace uso de Full-adders adicionales. Cuando se suman dos números binarios, cada columna genera un bit de suma y un bit de acarreo, la cual se añade inmediatamente a la columna de la izquierda.



Para sumar dos números binarios se hacen necesarios dos Full-adders por cada bit que tengan los números a sumar. La salida de acarreo de cada sumador se conecta con la entrada de acarreo del sumador de orden inmediatamente superior. Los bits menos significativos y son representados como A_1 y B_1 y los bits siguientes son de orden superior y son representados como A_2 y B_2

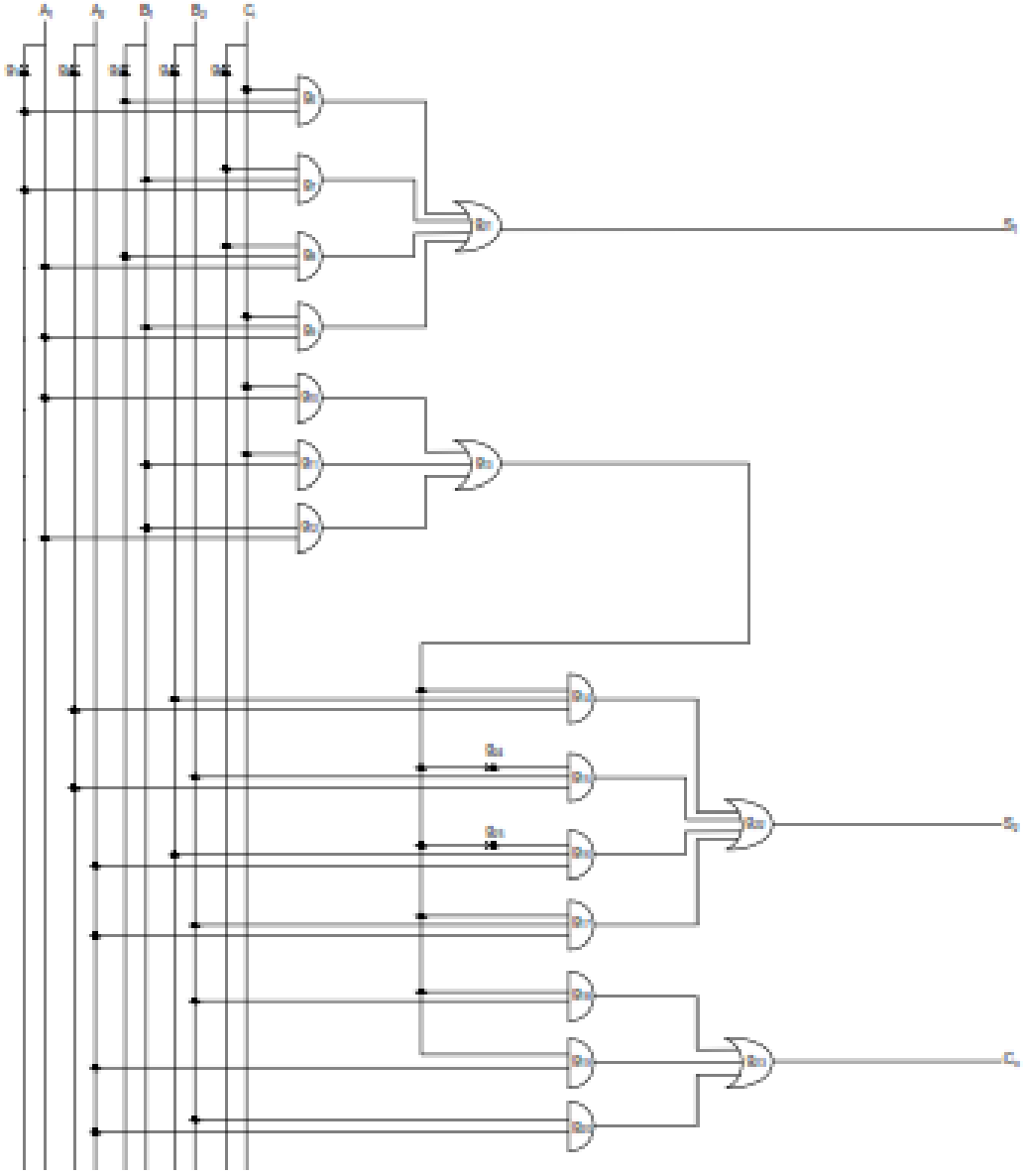


Figura12.2.1 Circuito del sumador paralelo de 2 bits

$$t_{Co} = 3t_{gate}\,; \quad t_{add} = 4t_{gate}$$

Obteniendo sus valores de medidas se tiene:

- **Tamaño del circuito:** $C_{U_{S1}}(f) = 8$
- **Tamaño del circuito:** $C_{U_{S2}}(f) = 12$
- **Tamaño del circuito** $C_{U_{Co}}(f) = 8$
- **Profundidad del circuito:** $D_{U_{S1}}(f) = 3$
- **Profundidad del circuito:** $D_{U_{S2}}(f) = 5$
- **Profundidad del circuito:** $D_{U_{Co}}(f) = 4$
- **Tamaño de fórmula:** $L_{U_{S1}}(f) = 7$
- **Tamaño de fórmula:** $L_{U_{S2}}(f) = 7$
- **Tamaño de fórmula:** $L_{U_{Co}}(f) = 6$
- **Fan-out:** $A_1 = 5\,, A_2 = 5\,,\ B_1 = 5\,,\ B_2 = 5,\ g_1 = 2\,,\ g_2 = 2\,,\ g_3 = 2\,,\ g_4 = 2,\ g_5 = 2,\ g_6 = 1,\ g_7 = 1\,,\ g_8 = 1\,,\ g_9 = 1\,,\ g_{10} = 1\,,\ g_{11} = 1\,,\ g_{12} = 1\,,\ g_{13} = 6,\ g_{14} = 1\,, g_{15} = 1\,, g_{16} = 1\,, g_{17} = 1\,, g_{18} = 1\,, g_{19} = 1\,,\ g_{20} = 1\,, g_{21} = 1, g_{22} = 1,\ g_{23} = 1\,,\ g_{24} = 1\,, g_{25} = 1$
- **Fan-in:** $g_1 = 1\,,\ g_2 = 1\,,\ g_3 = 1\,,\ g_4 = 1\,,\ g_5 = 1\,,\ g_6 = 3\,,\ g_7 = 3\,,\ g_8 = 3,\ g_9 = 3,\ g_{10} = 2,\ g_{11} = 2,\ g_{12} = 2,\ g_{13} = 3\,, g_{14} = 1\,, g_{15} = 1\,, g_{16} = 3\,, g_{17} = 3,\ g_{18} = 3\,, g_{19} = 3\,,\ g_{20} = 2\,, g_{21} = 2\,, g_{22} = 2\,, g_{23} = 4\,, g_{24} = 4\,, g_{25} = 3$

Ya obtenidas las medidas se hace uso del método de eliminación de trayectoria el cual consiste en poner una de las variables de entrada constante $A_2 = x_i = 0$ por lo tanto se ve afectado las salidas S_2 y la salida de acarreo C_0 y se obtienen las nuevas medidas:

- **Tamaño del circuito:** $C_{U_{S1}}(f) = 8$
- **Tamaño del circuito:** $C_{U_{S2}}(f) = 10$
- **Tamaño del circuito** $C_{U_{Co}}(f) = 6$
- **Profundidad del circuito:** $D_{U_{S1}}(f) = 3$
- **Profundidad del circuito:** $D_{U_{S2}}(f) = 5$
- **Profundidad del circuito:** $D_{U_{Co}}(f) = 4$
- **Tamaño de fórmula:** $L_{U_{S1}}(f) = 7$
- **Tamaño de fórmula:** $L_{U_{S2}}(f) = 5$
- **Tamaño de fórmula:** $L_{U_{Co}}(f) = 4$
- **Fan-out:** $A_1 = 5\,, A_2 = 5\,,\ B_1 = 5\,,\ B_2 = 5,\ g_1 = 2\,,\ g_2 = 2\,,\ g_3 = 2\,,\ g_4 = 2,\ g_5 = 2,\ g_6 = 1,\ g_7 = 1\,,\ g_8 = 1\,,\ g_9 = 1\,,\ g_{10} = 1\,,\ g_{11} = 1\,,\ g_{12} = 1\,,\ g_{13} = 6,\ g_{14} = 1\,, g_{15} = 1\,, g_{16} = 1\,, g_{17} = 1\,,\ g_{20} = 1\ \ g_{23} = 1\,,\ g_{24} = 1\,, g_{25} = 1$
- **Fan-in:** $g_1 = 1\,,\ g_2 = 1\,,\ g_3 = 1\,,\ g_4 = 1\,,\ g_5 = 1\,,\ g_6 = 3\,,\ g_7 = 3\,,\ g_8 = 3,\ g_9 = 3,\ g_{10} = 2,\ g_{11} = 2,\ g_{12} = 2,\ g_{13} = 3\,, g_{14} = 1\,, g_{15} = 1\,, g_{16} = 3\,, g_{17} = 3,\ g_{20} = 2\ \,, g_{23} = 4\,, g_{24} = 4\,, g_{25} = 3$

Como se observa al comparar las medidas obtenidas antes de aplicar el método de eliminación de trayectoria con respecto las medias después de aplicarlo se nota que los cambios se ven en el tamaño del circuito y en tamaño de fórmula y como el cambio de variable se aplicó para la entrada A_2 solo se observa el cambio en las salidas que dependen de dicha entrada

12.3 MULTIPICACIÓN EN NÚMEROS BINARIOS

La multiplicación de números binarios se hace de forma tradicional en donde el multiplicando se multiplica por cada bit del multiplicador, comenzando por el bit menos significativo. Cada uno de las multiplicaciones genera un producto parcial. Los productos parciales sucesivos tienen un desplazamiento hacia la izquierda. Y el resultado final se obtiene sumando los productos parciales.

Considerando una multiplicación de dos números de 2bits A_1A_2 y B_1B_2, en donde los bits del multiplicador son A_1A_2 y los del multiplicando son B_1B_2

$$\begin{array}{cccc} & & B_2 & B_1 \\ & & A_2 & A_1 \\ \hline & & A_1B_2 & A_1B_1 \\ & A_2B_2 & A_2B_1 & \\ \hline C_4 & C_3 & C_2 & C_1 \end{array}$$

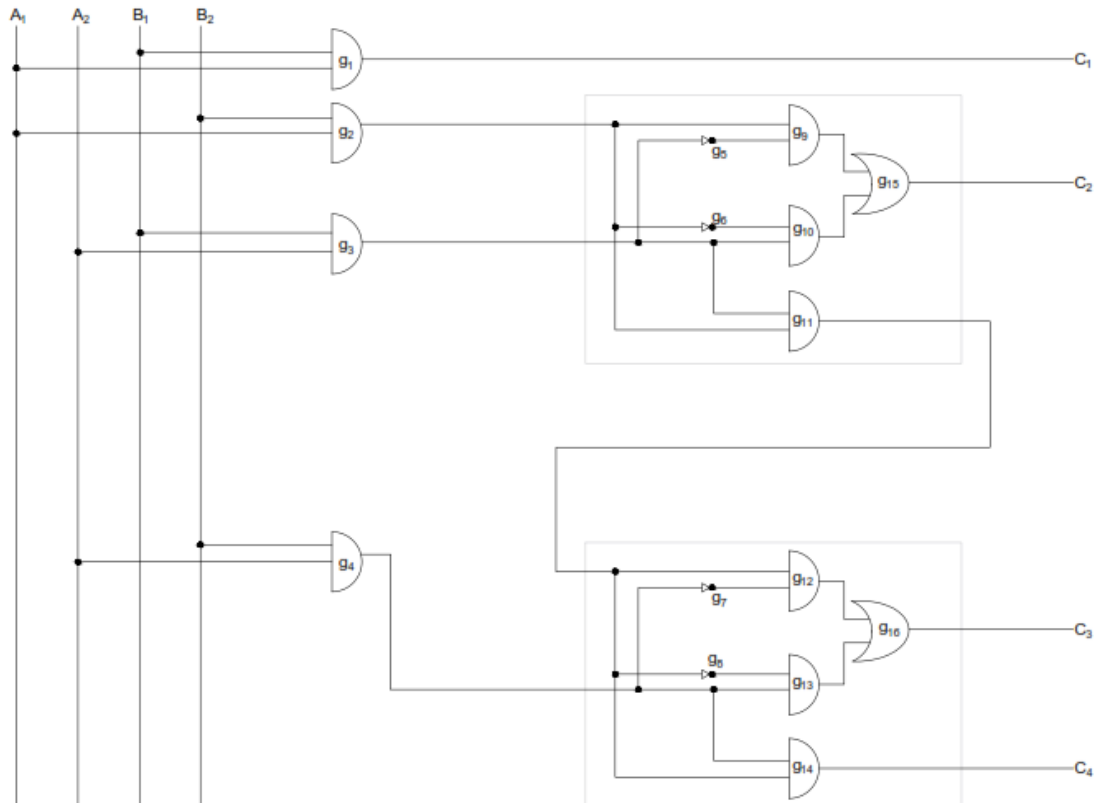


Figura 12.3.1 Circuito del método tradicional de la multiplicación para 2 bits

$$t_C = 4t_{gate}, \qquad t_{Co} = 3t_{gate}$$

Obteniendo los valores de las medidas de complejidad

- **Tamaño del circuito:** $C_{U_C}(f) = 14$
- **Tamaño del circuito** $C_{U_{C4}}(f) = 4$

- **Profundidad del circuito:** $D_{U_C}(f) = 4$
- **Profundidad del circuito:** $D_{U_{C_4}}(f) = 3$
- **Tamaño de fórmula:** $L_{U_C}(f) = 4$
- **Tamaño de fórmula:** $L_{U_{C_4}}(f) = 3$
- **Fan-out:** $A_1 = 2, A_2 = 2,\ B_1 = 2,\ B_2 = 2,\ g_1 = 1,\ g_2 = 2,\ g_3 = 2,\ g_4 = 2,\ g_5 = 1,\ g_6 = 1,\ g_7 = 1,\ g_8 = 1,\ g_9 = 1,\ g_{10} = 1,\ g_{11} = 2,\ g_{12} = 1,\ g_{13} = 1, g_{14} = 1, g_{15} = 1, g_{16} = 1$
- **Fan-in:** $g_1 = 2,\ g_2 = 2,\ g_3 = 2,\ g_4 = 2,\ g_5 = 1,\ g_6 = 1,\ g_7 = 1,\ g_8 = 1,\ g_9 = 2,\ g_{10} = 2,\ g_{11} = 2,\ g_{12} = 2, g_{13} = 2, g_{14} = 2, g_{15} = 2, g_{16} = 2$

Aplicando el método de eliminación de trayecto, el cual se hace convirtiendo una de las variables de entrada en 1.para la variable de entrada $A_1 = x_i = 1$ al aplicar este método para hallar el límite inferior se elimina automáticamente la trayectoria de A_1 hasta $C_1\ (g_1)$ y se ve afectada la trayectoria $(g_2 g_9)$convirtiendo en constante estas trayectorias. Con este cambio la ecuación cambia y se obtienen nuevas medidas

$$\begin{array}{cccc} & & B_2 & B_1 \\ & & A_2 & 1 \\ \hline & & B_2 & B_1 \\ & A_2B_2 & A_2B_1 & \\ \hline C_4 & C_3 & C_2 & C_1 \end{array}$$

- **Tamaño del circuito:** $C_{U_S}(f) = 13$
- **Profundidad del circuito:** $D_{U_S}(f) = 4$
- **Tamaño de fórmula:** $L_{U_S}(f) = 1$
- **Fan-out:** $A_1 = 2, A_2 = 2,\ B_1 = 2,\ B_2 = 2,\ g_3 = 2,\ g_4 = 2,\ g_5 = 1,\ g_6 = 1,\ g_7 = 1,\ g_8 = 1, g_9 = 1\ g_{10} = 1,\ g_{11} = 2,\ g_{12} = 1,\ g_{13} = 1, g_{14} = 1, g_{15} = 1, g_{16} = 1$
- **Fan-in:** $g_3 = 2,\ g_4 = 2,\ g_5 = 1,\ g_6 = 1,\ g_7 = 1,\ g_8 = 1,\ g_9 = 2,\ g_{10} = 2,\ g_{11} = 2,\ g_{12} = 2, g_{13} = 2, g_{14} = 2, g_{15} = 2, g_{16} = 2$

Como se observó anteriormente al aplicar este método se observa que al eliminar estas trayectorias se observa un cambio en el tamaño y el tamaño de fórmula del circuito. Pero no se observa ningún cambio en la profundidad del circuito dado que esta es medida por el trayecto que tenga más capas.

Como se pudo observar en los tres ejemplos anteriores en los que se aplicó los métodos para obtener el límite inferior de cada uno de ellos, se vio que los cambios se hacían notables en el tamaño de formula y el tamaño del circuito dado que ellos son dependientes de la cantidad de compuertas usadas ya sea de todo el circuito o de las usadas en la entrada del mismo.

CAPÍTULO 13. CONCLUSIONES

- La Máquina de Turing y un circuito digital son sumamente similares debido a que como la MT tiene como recursos un cabezal de lectura y una cinta de datos y un circuito tiene como recursos el tamaño, la profundidad y el tamaño de formula del circuito. Entonces si comparamos la utilidad entre los dos nos damos cuenta de que el cabezal de lectura de la MT y la profundidad del circuito hacen referencia al recurso de tiempo que usan los dos sistemas y al comparar la cinta de datos de la MT con el tamaño del circuito hacen referencia al recurso de la memoria.

- Inicialmente cuando se habló de la medición de recursos de para la complejidad se planteó que se necesitaba obtener el límite superior para conocer cuál debía ser el límite y no superarlos. Pero a medida que se continuo investigando se conoció que si se refiera a los recursos un circuito los diseñadores están más interesados en conocer el limite inferir de los recurso que se necesita para completar una tarea dada.

- El límite superior en los circuitos aritmético solo sería útil para circuitos la cuales tienen formulas muy pequeñas generalmente para circuitos de dos bits de entrada y dos bits de salida.

- Cuando se va a obtener los recursos usado por una función si se habla de resolverlo por medio de Software se requiere que se obtenga el límite superior; pero si por el contrario para Hardware se necesita el límite inferior.

- Al aplicar cualquier método para obtener el límite inferior es necesario que los circuitos sean monótonos.

- La profundidad del circuito es la medida en tiempo paralelo necesario para calcular una función

BIBLIOGRAFÍA

[1] R. J. T. N. S. W. y G. L. M. , Sistemas digitales principios y aplicaciones, 10 ed., México: Pearson Editorial, 2007.

[2] R. BRENA, Autómatas y lenguajes un enfoque de diseño, Monterrey: Tec de Monterrey, Verano 2003.

[3] T. L. FLOYD, Fundamentos de sistemas digitales, 9 ed., Madrid: PEARSON EDUCACIÓN S.A., 2006, p. 1024.

[4] N. CHOMSKY, Three models for the description of language, Massachusetts: IRE Trans. on Information Theory, 1956, pp. 113-124.

[5] S. BALARI, TEORIA DE LENGUAJES FORMALES, Barcelona: Universidad Autónoma de Barcelona , 2014.

[6] J. E. SAVAGE, Models Of Computacions, Unidersidad de Brown , 1998.

[7] R. KÁLOVIC, Mathematical Foundations Of Computer Science 2009, Slovakia : Damian Niwinski, 2009.

[8] J. V. LEEUWEN, Handbook of Theoretical Computer Science, Volumen A, Algorithms and Complexity, Amsterdam: Managing, 1990.

[9] E. A. LEE y A. SANGIOVANNI-VICENTELLI, «A framework for comparing models of comptations.,» *IEEE Transaction on computer-aided desing of integrated circuits and systems ,* vol. 17, nº 12, pp. 1217-1229, 1998/12.

[10] N. MARGOLUS, «Physics-like model of computation,» Cambridge Massachusetts, 1984.

[11] A. BORODIN y J. E. HOPCROFT, «Routing, merging, and souting on parallel models of computation,» *Journal of computer an system science ,* vol. 30, nº 1, pp. 130-145, 1985.

[12] M. SIPSER, Introduction to the Theory of Computation, 3 ed., CENGAGE Learning, 2013.

[13] D. C. KOZEN, Automata and Computability, reprint of 1st ed., Springer, 2013.

[14] J. E. HOPCROFT, R. MOTWANI y J. D. ULLMAN, Introduction to Automata Theory, Languages, and Computation, 2nd ed., Pearson Education, 1939.

[15] M. HERNANDEZ, Models of Computation, Springer, 2009.

[16] A. RALSTON, E. D. REILLY y D. HEMMENDINGER, Encyclopedia of Computer Science, John Wiley and Sons Ltd., 2003.

[17] J. ERICKSON, Models of Computation, 2015.