Simulación de modelos

Erick Hernandez Navarrete

31 de octubre de 2020

Índice general

Capítulo 1

Decidibilidad en los modelos

1.1. Introducción

En este documento se expondrán las nociones de decibilidad, complejidad en ambos modelos en el mundo de máquinas de Turing y en el mundo distribuido, así como también se hará la formalización de la noción de simulación de modelos computacionales, demostrando que el modelo A de la máquina de Turing es equivalente en poder al modelo distribuido B, en particular al modelo **LOCAL**

1.2. Decidibilidad del modelo máquinas de Turing

1.2.1. Cadenas y lenguajes

Ahora, se definirán los bloques constructores de ciencias de la computación, es decir las cadenas de carácteres. Con el propósito de definir lo siguiente:

Definition 1. *Decimos que un alfabeto es cualquier conjunto X tal que $X! = \emptyset$*

Una vez que se define el concepto de alfabeto, los elementos del lenguaje son los símbolos del alfabeto. Se estipulará que las letras mayúsculas se usarán para los alfabetos y letras minúsculas para los símbolos. En virtud de ello se pude enuciar algunos ejemplos de alfabetos:

- $\Theta_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $\Theta_2 = \{a, b, c, d, e, f, g, h, i, j, k, l\}$
- $\Omega = 0, 1, x, y, z, t$

Lo siguiente es definir la noción de una cadena sobre un alfabeto, que para los fines de este estudio son los bloques constructores del mismo, y que son, al mismo tiempo, en el modelo de máquinas de Turing las entradas de instancias de este modelo, lo que dará la noción de cómputo.

Definition 2. *Una **cadena** sobre un alfabeto Σ es una secuencia finita de símbolos del alfabeto, que es usualmente escrito cada símbolo uno sobre otro y sin separación por comas.*

Una vez que se tiene la noción de cadena sobre un alfabeto, se puede dar unos cuantos ejemplos:

- Si $\Sigma_1 = \{a, b\}$, entonces la cadena $abbaba$ es una cadena sobre Σ_1
- Si $\Sigma_2 = \{2, 3, 5\}$, entonces la cadena 32225 es una cadena sobre Σ_2

Entonces ahora que se definió el concepto cadena sobre un alfabeto podemos definir atributos a las cadenas definidas sobre un alfabeto Σ,

Definition 3. *Sea w una cadena sobre un alfabeto Σ,dicha cadena tiene* ***longitud****, y se denotorá como $|w|$, y asi se concluirá que es el número de símbolos que contiene.*

Entonces con base a esa definición, se denominará vacía a la cadena tal que su longitud es cero, y la distinguiremos como ϵ.
Se puede escribir a una cadena $w = w_1, \dots w_n$, sobre un alfabeto Σ si cada $w_i \in \Sigma$; A continuación se definirán ciertas operaciones o atributos asociados a estas estructuras de datos.

Definition 4. *Sea una cadena w, se asociará a la cadena w la cadena reversa denotada como w^R y con la notación, si $w = w_1, \dots w_n$, entonces $w^R = w_n, \dots, w_1$.*

También se dará la noción de sub-cadena, la cual enunciaremos de la siguiente manera:

Definition 5. *Sean z, w cadenas, se puede decir que z es una sub-cadena de w si aparece de manera consecutiva dentro de w.*

Ejemplos de subcadenas son:

- abc es una sub-cadena de la cadena $abcdedfg$
- $cdcd$ es una subcadena de la cadena $cdasdcadccdcd$

También se definirá una operación entre dos cadenas, la cual es la concatenación, entonces se define formalmente de la siguiente manera:

Definition 6. *Sean w_1 y w_2 dos cadenas finitas, esto es:*
$\exists n$, $\exists m$ tal que $|w_1| = n$ y $|w_2| = m$ entonces la cadena $r = w_1 w_2$, es el resultado de agregar la cadena w_2 al final de la cadena w_1.
En notacion es: $concat(w_1, w_2) = w_1 w_2$

Entonces con esta definición de operación se hará el equivalente en potencia en matemáticas, formalmente lo podemos decir:

Definition 7. *Sea x una cadena sobre un alfabeto Σ, entonces concatenar dicha cadena $m-veces$, con $m \in$ se escribirá así:*
$x \dots x = x^m$

Finalmente se tiene la noción universo de las cadenas para algún alfabeto, que se formalizará de la siguiente manera:

Definition 8. *Sea un alfabeto Σ, se dice que un lenguaje es un conjunto de cadenas sobre el alfabeto Σ, i.e*

$$L = \{w : w \text{ es una cadena sobre} \Sigma\} \tag{1.1}$$

Entonces, una vez que se tiene la definición de lenguaje, lo siguiente es presentar el modelo de máquina de Turing, ya que en este contexto los problemas se modelarán en términos de lenguaje, con la noción anteriormente anunciada.

1.2.2. Elementos del modelo

Ahora, se dará la definición del modelo formal de computo, a saber el modelo de máquina de Turing, para ello se tendrá una presentación intuitiva del modelo.
El modelo de máquina de Turing es a grandes razgos un modelo con cierta robustez; lo cual quiere decir que se tendrá la posiblidad de resolver una cantidad de problemas, pero con ciertas limitantes.

Este modelo fue propuesto por primera vez en 1936 por el matemático **Alan Turing**, similares a otros modelos de cómputo que no se presentará en esta tesis, como son solo por decir: autómatas finitos. El modelo de máquina de Turing usa una cinta infinita como su memoria ilimitada, además tine el cabezal, que es la que permité leer y escribir símbolos además de moverse a lo largo de la cinta.
Inicialmente la cinta contiene únicamente como entrada a la cadena, y lo demás es blanco. Si la máquina necesita almacenar información, esta tendrá que ser escrita en la cinta.
Para leer la información que ha sido escrita en la cinta esta tendrá que mover su cabezal sobre ella.
La máquina continurá computando hasta que tenga una salida, lo que se traduce a que tendrá una sucesión de pasos como los que se describieron anteriormente, hasta que se dé la definición formal de la palabra cómputo.

Las salidas: aceptación, rechazo, se obtienen una vez que se ha entrado en los estados de aceptacion, rechazo, por otro lado si no entrá en alguno de los estados que se mencionaron anteriormente, la máquina continurá para siempre, i.e ésta nunca terminará de hacer su cómputo.
Esta es la presentación intuitiva de este modelo, entonces lo que sigue es definir de manera formal los elementos del modelo,

Definition 9. *Una máquina de Turing is una 7-tupla,* $(Q, \Sigma, \Lambda, \delta, q_0, q_{accept}, q_{reject})$ *donde* Q, Σ, Λ *son conjuntos finitos.*

1. Q *es el conjunto de estados,*
2. Σ *es el alfabeto de entrada que no contiene el símbolo blanco* $\sqcup$
3. Λ *es el alfabeto de la cinta donde* $\sqcup \in \Lambda$ *y además* $\Sigma \subseteq \Lambda$,
4. $\delta : Q \times \Lambda \to Q \times \Lambda \times \{L, R, S\}$ *es la función de transición*
5. q_0 *es el estado de iniciación,*
6. q_{accept} *es el estado de aceptacion*
7. q_{reject} *es el estado de rechazo donde* $q_{accept} \neq q_{reject}$,

Aquí los símbolos L, R, S son el imperativo para el movimiento del cabezal ***Left****,* ***Right*** *y* ***Stay***

Una vez que se tiene la definición se presentará como es el cómputo:
Inicialmente, se tendrá como entrada una cadena $w = w_1 \ldots w_n$ en Σ, en a lo más n cuadros a la izquierda y el resto de la cinta esta en blanco.Ésto quiere decir que están rellenados con símbolos blancos, y la manera de identificar el final de la entrada w es marcado por el primer símbolo en blanco en la cinta, ya que el alfabeto Σ no contiene dicho símbolo.

Luego, una vez que inicia la máquina M con la entrada w, el cómputo estará gobernado por la función δ. Algo que se puede observar es que si en algún momento del cómputo el cabezal intenta moverse a la izquierda del lado izquierdo extremo, éste se quedará en el mismo lugar, aunque la función de transición indique un movimiento del cabezal L.

Finalmente, el cómputo continuará hastá que entre en el estado de q_{accept} ó q_{reject}, si no sucede lo anterior M continuará para siempre. Una vez presentado el modelo y una visión intuitiva del cómputo en el mismo, se adentrará a la noción de **decidible**, que es la noción a la que se quiere converger en este modelo para hacer la conexión con el modelo de cómputo distribuido.

1.2.3. Decidibilidad

Ahora la noción de **configuración** es lo que nos permitirá formalizar el movimiento del cabezal de la máquina, que esto en esencia será la formalización del cómputo en este modelo.
La manera en que M hace el compúto, de manera intuitiva se puede decir que ocurren ciertos eventos, a saber:

- Cambio en el estado actual
- Cambio en el contenido de la cinta
- Cambio en la localidad del cabezal

Estos eventos se constituirán al definir el término configuración. Los elementos serán en esencia tomarse el estado actual, digamos q_a, el contenido actual de la cinta, por ejemplo: uv con u, v cadenas para el alfabeto Σ y la localidad actual de la cinta, que será el primer símbolo de la cadena v.
Esto se puede plantear en la triada:

$$C = (u, q, v) \tag{1.2}$$

Con u, v cadenas y q es el estado actual, y la posición actual es el primer símbolo de la cadena v, en este sentido se puede escribir de la siguiente manera esta noción:

$$uqv \tag{1.3}$$

Para enfatizar que en ese momento del compúto, el cabezal esta en la localidad del primer símbolo de la cadena v, se coloca a el estado actual q entre la cadena u y v; esto da un significado formal a la notación del concepto de configuración.

Entonces esta noción es la que va a dar el paso de formalizar el compúto, por lo cual de manera intuitivamente se va a tener una secuencia de configuraciones:

$$C_0 \ldots C_n \in \mathbb{N} \tag{1.4}$$

Pero este proceso se definirá entre dos estados a priorí C_i, C_j; entonces esta noción permitirá definir lo siguiente:

Definition 10. *Se dice que la* ***configuración*** C_j *le sigue a* C_k *si: Se toman las cadenas* u, v *de* Γ*, y también tomamos* $a, b \in \Sigma$*. entonces se concluye que: si pasa* $\delta(q_k, b) = (q_j, c, L)$*, entonces* $uacq_jv$ *le sigue de* uaq_kbv

La anterior noción se esta definiendo con la función δ, que es la que esta formalizando esta noción. Como se menciono anteriormente, se estipula que las secuencias de configuraciónes, entonces se tendrá una configuración inicial.

Definition 11. *Sea* M *una máquina de Turing,* w *una cadena, tomamos a la configuracion inicial* C_0*, y se representa con base a lo anterior como* q_0w

El significado de la notación de la configuración inicial es que esta en el estado cero q_0 y el cabezal esta en el primera localidad de lado izquierdo de la cinta. Ahora las configuraciones que intuitivamente son las candidatas para que sean las configuraciones finales si es que la máquina termina en algun momento de este proceso; pero por el momento se llamará a esas configuraciones:

$$q_{accept}, q_{reject} \tag{1.5}$$

A estas configuraciones se les nombrarán como configuraciones de paro, ya que están definidas en términos del estado en el que entran; que son los estado de paro y de aceptación respectivamente.
Entonces se formalizará que una máquina de Turing se detiene si entra en los estados $q_{accept0}, q_{reject}$ y con base a esto se hará una generalización de la definición de la función delta, tomando un conjunto de estados en el que no esta los estado de detención
Una vez aclarado esto, se tienen los elementos para definir que:
Dada una máquina M acepta a una cadena arbitraria w.

Definition 12. *Sea una cadena* w *y una máquina de Turing* M*,*
entonces se dice que M *acepta a la cadena* w *si:*

1. C_1 *es la configuración inicial con entrada* w
2. *cada* C_{i+1} *le sigue de* C_i
3. *existe un* $k \in$ *tal que* C_k *es una configuración de aceptación*

Esto esta formalizando la noción de que una máquina de Turing **acepta** a una cadena w y lo que se observa es que los problemas que se plantean, se escriben en términos del lenguaje, es decir, el sentido formal de lenguaje; que son los conjuntos de cadenas formadas sobre un alfabeto dado Σ.

En este sentido, dada una máquina de Turing M,se puede tener la çolecciónño"de todas las cadenas que son aceptadas por M, que con base a la definición de lenguaje en el mundo de máquinas de Turing esto es el conjunto de cadenas con el atributo de que son aceptadas por M, sobre un alfabeto Σ.
Ahora se formalizara lo anterior de la siguiente manera:

Definition 13. *Sea M una máquina de Turing, al conjunto de cadenas w que son aceptadas por M sobre el alfabeto Σ, se llamarán el lenguaje que **reconoce** M, se escribirá como $L(M)$.*

Ahora con estas definiciones, se tendrá la siguiente noción:
Sea L un lenguaje sobre un alfabeto Σ, se estipulará en términos de la existencia de una máquina de Turing que lo hace reconocible; así se formalizará de la siguiente manera:

Definition 14. *Sea L un lenguaje, se dice que es Turing-reconocible si: existe una máquina de Turing que lo hace Turing-reconocible.*

Remark. *Cuando se inicia una máquina de Turing con entrada w, pasan tres posibles eventos:*

- *acepta,*
- *rechaza,*
- *nunca se detiene.*

En este caso, se tomará solamente máquinas de Turing que se detienen en algun momento en la ejucución para todas sus entradas, por lo tanto en términos formales, dichas máquinas siempre se dentendrán en algún momento de la ejecución de M, para $\forall w$ entrada. Se formalizará lo anterior con una definición.

Definition 15. *Sea M máquina de Turing tal que $\forall w$ entrada, si esta se detiene en algún momento de la ejecución, entonces dicha máquina de Turing tiene el atributo de ser **decidible***
*Donde w es una cadena sobre el alfabato Σ Es decir, siempre entrán en su estado de **aceptación** o en su estado de **rechazo** en algún momento de la ejecución.*

Y por el lado de la noción del lenguaje definimos lo siguiente:

Definition 16. *Se dicé que un lenguaje es **decidible** si existe un máquina de Turing que lo **decide**. i.e que la máquina tiene el atributo de ser decidible.*

Una vez que se tiene el concepto de máquina de Turing, daremos un ejemplo de un lenguaje que es decidible, i.e expondremos una máquina de turing **decidible**.

Example 1. *Tomamos el siguiente lenguaje:*

$$A = \{0^{2n} : n >= 0\} \tag{1.6}$$

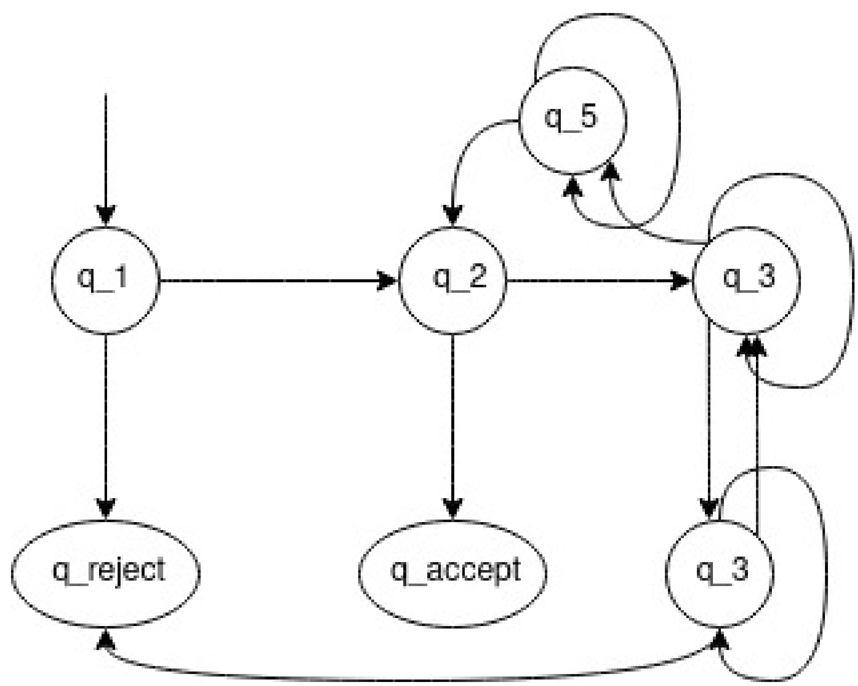
Es el lenguaje que consta de cadenas de 0's tal que su longitud es una potencia de 2.

Entonces se afirma que M_2 es un lenguaje decidible. Entonces se puede hacer las instrucciones (alto nivel) para M_2 como sigue:

1. Recorrer de izquierda a derecha a través de la cinta, tachando uno de cada dos ceros
2. Si en la etapa 1, la cinta contenia un solo 0, entonces **acepta**
3. Si en la etapa 1, la cinta contenía mas de un 0 y además un número impar, mayor a 1, **rechaza**
4. Regresa el cabezal al lado izquierdo al final de la cinta
5. Ve a la etapa 1

Una vez que se tiene la descripción de la máquina de Turing a un alto nivel, se puede hacer un puente a un bajo nivel, que es formalmente describir la máquina con sus elementos que la definen para una entrada w, el cual nuestro puente será en describir la función δ vía un diagrama de estados En la etapa 1 de M_2, corta el número de ceros a la mitad. Mientras la máquina barre tráves de la cinta en la etapa 1, lleva la cuenta de 0's vistos si es par o impar.
Si el número es impar mayor a 1, entonces el número original de 0's en la entrada no puede ser una potencia de 2. Entonces la máquina **rechaza** en esa instancia. Si el número de 0's visto es 1, entonces el número original tiené que ser una potencia de 2, entonces en ese caso la máquina **acepta**. Sea $M_2 = (Q, \Sigma, \Lambda, \delta, q_1, q_{accept}, q_{reject})$:

1. $Q = \{q_1, q_2, \ldots, q_5, q_{accept}, q_{reject},$
2. $\Sigma = \{0\}$ and
3. $\Lambda = \{0, x, \sqcup\}$.
4. Se describirá a δ con un diagrama de estado.
5. Los estados de inicio, aceptación y rechazo son $q_1, q_{accept}, q_{reject}$ respectivamente.



En este diagrama, la etiqueta $0 \rightarrow \sqcup, R$ que muestra en la transición del estado q_1 al estado q_2, tiene la semántica de la definición de la función δ, que en otras palabras se describe como si:
en el estado q_1 con el cabezal leyendo 0, la máquina va al estado q_2, escribe $\sqcup$ y mueve el cabezal a la derecha.
Lo que en términos formales significa que: $\delta(q_1, 0) = (q_2, \sqcup, R)$. Otra escritura que se usará para la notación en este ejemplo más compacta, es denotar: $0 \leftarrow R$, que la semántica es que se hace la transición del estado q_3 al estado q_4, que es la etiqueta que se aprecia en el diagrama de estados, y ademas significa que la máquina se mueve a la derecha en el momento en el que lee un 0 en el estado q_3, pero que no altera la cinta(hace una operación escritura), entonces $\delta(q_3), 0) = (q_4, 0, R)$ en términos del mapeo que se esta generando.
Esta máquina inicia escribiendo un símbolo blanco sobre el último 0 de lado izquierdo. En particular esto sirve para delimitar el final de la cinta con el símbolo blanco $\sqcup$, en esta particular máquina de Turing.
Ahora haremos una ejecución de esta maquina, M_2 con la entrada $w = 0000$, con lo cual se escribirá la función transición δ con la semántica y escritura de las configuraciones.

La siguiente secuencia es la ejecución de M_2 con la entrada en particular $w = 0000$, Se leé hacía abajo de las columnas de izquierda a derecha.

$$
\begin{array}{ccc}
q_1 0000 & \sqcup q_5 x 0 x \sqcup & \sqcup x q_5 x x \sqcup \\
\sqcup q_2 000 & q_5 \sqcup x 0 x \sqcup & \sqcup q_5 x x x \sqcup \\
\sqcup x q_3 00 & \sqcup q_2 x 0 x \sqcup & q_5 \sqcup x x x \sqcup \\
\sqcup x 0 q_4 0 & \sqcup x q_2 0 x \sqcup & \sqcup q_2 x x x \sqcup \\
\sqcup x 0 x q_3 \sqcup & \sqcup x x q_3 x \sqcup & \sqcup x q_2 x x \sqcup \\
\sqcup x 0 q_5 x \sqcup & \sqcup x x x q_3 \sqcup & \sqcup x x q_2 x \sqcup \\
\sqcup x q_5 0 x \sqcup & \sqcup x x q_5 x \sqcup & \sqcup x x x q_2 \sqcup \\
 & & \sqcup x x x \sqcup q_{accept}
\end{array}
$$

1.3. Decidibilidad en el modelo de cómputo distribuido LOCAL

Ahora lo que se hará es definir el modelo de cómputo distribuido de manera formal y se tomará un modelo en particular en la que se enfocará nuestro estudio, para posteriormente estudiar la noción de decibilidad en dicho modelo.

En este modelo presentaremos las partes que lo conforman, todo ello de manera formal para ello haremos uso de elementos de cómputo teórico, a priori esto se podrá ver como una gráfica donde cada nodo se puede representar como una máquina de estados.

1.3.1. Presentación del modelo de computo distribuido

Presentación del protocolo de comunicación

Este modelo tendrá una capa de abstracción de comunicación, asi como su respectiva capa de computación de la siguiente manera.

Capa de comunicación

El modelo de comunicación consiste en una red de comunicación 1-1 que será descrita en términos formales por una gráfica conexa, no dirigida $G = (V, E)$, donde los vertices $V = \{v_1, \dots v_n\}$, operando entre ellos. Inicialmente, se considerará identificadores únicos asignados a los procesos de la gráfica G. Concretamente consideramos a estos identificadores de un conjunto ordenado de enteros, de la siguiente manera:

$$S = \{s_1, \dots, s_n\} \; donde: \; s_i < s_{i+1} \; \forall i \geq 1 \tag{1.7}$$

Entonces con esta notación, una ID-asignación es un mapeo: $ID : V \to S$, entonces se refiere a su identificador con la siguiente notación: $ID(v)$.
Se tiene que la comunicación se lleva acabo de la siguiente manera:

Cada vértice tendrá asociado el numero de puertos, como el $deg_G(v)$, luego en este sentido decimos que el conjunto de aristas adyacentes al vertíce contiene exactamente $deg_G(v)$, donde cada arista esta conectado en un puerto de v.
Se denotará que a cada arista (u, v) le corresponde la pareja $((u, i), (v, j))$, donde $1 \leq i \leq deg_G(u)$ y $1 \leq j \leq deg_G(v)$, con el siguiente contexto :
un canal de comunicación que se conecta en el puerto i de u con el puerto j de v.
El vértice u envía(ejecuta la operación $send()$) un mensaje a sus vecinos v, cargando el mensaje en un puerto apropiado, digamos i. Este mensaje es recibido($deliver()$) por v, a tráves del puerto j.

Capa de Computación

Una vez que se tiene la capa de comunicación, se presentará la capa formal del modelo de cómputo.
El modelo estara controlado por un algoritmo Π, que estará compuesto de protocolos(algoritmos) $\Pi_1, \dots, \Pi_n$, donde cada Π_i residirá en su correspondiente v_i.

Remark. *Hasta ahora hemos hablado a secas de los vértices, pero se dará un contexto de cómputo en términos de **procesos**, lo cual significa que es una entidad de cómputo, y entonces a cada v_i se nombrará como el proceso p_i.*
Con esta convención se dirá que cada protocolo(algoritmo local) Π_i residirá en su respectivo proceso: p_i.

Remark. *Se observa que se modelará a cada Π_i como una máquina de estado para $\forall i$ con su correspodiente conjunto de estados estado Q_i conteniendo en particular a su estado inicial q_{0i}, así como sus estados de aceptación y rechazo: $q_{accept}, q_{reject} \in Q_i$ tal que en cualquier momento dado el proceso p_i esta en el estado q_i de Q_i. Mas aún se interpretará a cada Π_i como en una máquina de Turing, equipada con operaciones de envío y recepción de **mensajes**.*

Por otro lado en la capa de comunicación, se tiene el siguiente esquema:

Definition 17. *Se definirá un mensaje MSG como la información local que será enviada del proceso v al proceso u, por medio del canal $((v, i), (u, j))$, donde el proceso v tiene el atributo de la*

operación send(MSG), y el vértice v ejecuta una operación deliver(MSG), para que finalmente el proceso v ejecute una operación compute().
Se dirá a manera de observación que el tamaño de la información del mensaje MSG, es $O(\log n)$ bits.

En cualquier momento y en cualquier canal de comunicación $e_i = (u, v)$ está en algun estado $\overline{q}_i$ del conjunto de estados $\overline{Q}_i$ el estado $\overline{q}_i$ esta compuesto de dos componentes denotadas de la siguiente manera: $\overline{q}_{u \leftarrow v}$ y $\overline{q}_{v \leftarrow v}$ una por cada direccion del canal de comunicación.

Se denotará como M a la colección de todos los posibles mensajes que se pueden enviar de un proceso a otro en toda ejecución del algoritmo, cada uno de los dos componentes $\overline{q}_{u \leftarrow v}$ es un elemento de $M \cup \lambda$, $\overline{q}_{u \leftarrow v} = MSG \in M$ significa que ahora el mensaje MSG está en transición de u a v, y se denotara que $\overline{q}_{u \leftarrow v} = \lambda$ para representar el hecho de que el canal actual esta vacio en esa dirección. En el inicio del cómputo todos los procesos están en el estado inicial $q_{0,i}$ $\forall i$ y todos los canales de comunicación estan vacios. Es decir se escribe como: $\overline{q}_{i,0} =< \lambda, \lambda >$

Ejecución de un algoritmo en este modelo

La ejecución del algoritmo en este ambiente consiste de **eventos**, ocurriendo en diversos lugares de la red y afectando a los procesos involucrados. Se dirá que un **paso computacional** es una operación como máquina de Turing del proceso p. Los eventos puede ser del tipo:

- Computacional: Representando un paso en un procesador
- Comunicación: Representando la entrega o la recepción de un mensaje

Donde cada evento de comunicación se interpreta como:
$SEND(i, j, MSG)$ o $DELIVER(i, j, MSG)$ para algún mensaje MSG. Entonces enlistando los eventos:

1. Evento $COMPUTE(i)$: El proceso v_i ejecuta una operación interna, basado en su estado local y posiblemente mute su estado local
2. Evento $SEND(i, j, MSG)$: El proceso v_i envia de salida un mensaje MSG en algún canal de comunicación link e_l con destino al proceso v_j
3. Evento $DELIVER(i, j, MSG)$: El mensaje MSG originado de un proceso v_i que es enviado por el canal de comunicación e_l es entregado en la entrada del destino v_j

Entonces la computación en un sistema distribuido se pensará de la siguiente manera:
como una secuencia de configuraciones, capturando el estado actual de los procesos y los canales de comunicación.
Cada evento cambia de estado para algún procesador v_i y posiblemente también para un canal de comunicación y eso cambiará la configuración del sistema. En términos formales se pensará de la siguiente manera:

Definition 18. *Una **configuración** es una tupla $(q_1, \ldots, q_n, \overline{q}_1, \ldots, \overline{q}_m)$, donde $q_i, \overline{q}_j$ es el estado del procesador p_i y del canal de comunicación e_j respectivamente y la configuración inicial es:*

$$q_{0,1}, \ldots, q_{0,n}, \overline{q}_{0,1}, \cdots \overline{q}_{0,m} \tag{1.8}$$

Remark. *Como se imagina de manera intuitiva a cada uno de los procesos como una máquina de Turing, entonces una vez que uno de los procesos entra en alguno de los estados:* q_{accept}, q_{reject}*, el proceso ya no muta de estado, pero sus operaciones de envío y recepción de mensajes siguen activos, i.e quedan activos las operaciones de* $send(), receive()$*.*

Luego se modelará la computación del algoritmo como una (posible) infinita secuencia de configuraciones alternadamente con eventos.

Definition 19. *La ejecución de un algoritmo* Π *en una gráfica con cierta topología* G*, con una entrada inicial* I *en los procesos es denotado como* $\kappa_{\Pi(G,I)}$*. Formalmente, una* ***ejecución*** *es una secuencia de la forma:*

$$\kappa = (C_0, \rho_1, C_1, \rho_2, C_2, \ldots) \tag{1.9}$$

donde cada C_k *es una configuración y cada* ρ_j *es un evento, y en particular* C_0 *denota la configuración inicial.*

Se impondrá ciertas restricciones en las ejecuciones del algoritmo, pero por el momento se definirá normalmente el concepto en términos de las ejecuciones para algún algoritmo Π.
Entonces lo anterior nos da pauta a definir lo siguiente:

Definition 20. *Se dirá que un* ***modelo*** *es un subconjunto de esas posibles ejecuciones*

Con la definición anterior, se tomará un modelo en particular con una propiedad en particular, a saber el que tiene una estructura de rondas.

Definition 21. *Se dirá que un* ***modelo*** *tiene el atributo de* ***rondas****: Si las ejecuciones tienen estructura de* ***rondas****, es decir:*

1. *Cada proceso* p *ejecuta* ***send*** *a todos sus vecinos,* ***deliver*** *de todos sus vecinos y finalmente* ***compute****,*

2. *Cada proceso ejecuta su* r*-ésima ronda si todos los procesos ejecutaron su* $r-1$ *ronda.*

La definición anterior permitirá definir el siguiente modelo:

Definition 22. *Llamaremos* ***LOCAL*** *al modelo que tenga el atributo de* ***rondas***

Se observá que el punto 2 de la definición de rondas, es la que da el carácter de síncronía en la ejecución.

Ejemplo de un algoritmo distribuido

Se presentará un ejemplo desarrollado en el proceso de esta tesis.

Dado una red de procesadores con cierta topología, a saber la topología de un arbol, donde cada nodo tiene alojado en memoría un entero n, entonces el problema a solucionar es encontrar el elemento mas repetido en toda la red, i.e encontrar estadísticamente el modo de la red; para eso debemos implementar un protocolo de comunicación en el Arbol para determinar dicho valor. Que en términos formales es diseñar un algoritmo que sera una tupla de n protocolos los cuales se ejecutarán en el Arbol para solucionar dicho problema.

Definition 23. *Conceptualmente se tendrá los siguientes tipos de mensajes que se enviarán en el protocolo que se enuciará mas adelante:*

- *Mensajes del tipo 1:* $M(\uparrow, A_{i1}, A_{i2})$
- *Mensajes del tipo 2:* $M(\uparrow, (x_{modo}), max(A_{i2}))$

Donde A_{i1} es un arreglo de los valores locales de los nodos del sub-arbol enraizado en el nodo p_i, y A_{i2} es un arreglo de las frecuencias de los valores locales en A_{i1}

Definition 24. *Sean* $M \uparrow, A_{i1}, A_{i2})$*,* $M(\uparrow, A_{k1}, A_{k2})$ *mensajes del tipo 1, entonces se definirá la operación* $Sum(M(\uparrow, A_{i1}, A_{i2}), M(\uparrow, A_{k1}, A_{k2}))$ *y la definimos de la siguiente manera:*

1. *Sea* x *en* A_{i1}*, si* x *es tal que es distinto a todo* z *en* A_{j1} *entonces*
   $A_{r1} = A_{i1}[y] \star A_{k1}$ *y* $A_{r2} = A_{i2}[y] \star A_{k2}$
   donde $\star$ *denota la concatenación del arreglo formado por el elemento* $x = A_{i1}[y]$ *con* A_{k1} *y también denota la concatenación del elemento* $f_x = A_{i2}[y]$ *con el arreglo* A_{k2}*, es decir estamos generando un arreglo que contiene a los que estamos operando bajo* $\star$
2. *Sea* x *en* A_{i1}*, si es tal que* $x = A_{i1}[y] = A_{k1}[y']$ *entonces* $A_{r1}[y''] = A_{i1}[y] = A_{i2}[y']$ *y* $A_{r2}[y''] = A_{i2}[y] + A_{k2}[y']$

Y denotamos a $Sum(M(\uparrow, A_{i1}, A_{i2}), M(\uparrow, A_{k1}, A_{k2})) = M(\uparrow, A_{r1}, A_{r2})$

A continuación se mostrará el código del algoritmo.

Algorithm 1 First-dis(T_{r_0}, ID, x_i)

```
A_first(p_i, ID, x_i ∈ ℕ)
if  p_i es una hoja then
   A_r1 = (x_{p_i})
   A_r2 = (1)
   send M_{p_i} = M(↑, A_r1, A_r2) a su padre
else
   wait M_1, ..., M_n mensajes de sus hijos
   M_j = M(↑, A_j1, A_j2)
   for all l, s ∈ {1, ..., n+1} con l ≠ s do
      Sum(M(↑, A_l1, A_l2), M(↑, A_s1, A_s2))
   end for
   M(↑, A_r1, A_r2) = Sum(M(↑, A_11, A_12), ..., M(↑, A_n1, A_n2))
   A_{p_i}1 = A_r1 // Es el array después de las operaciones de los mensajes de sus hijos
   A_{p_i}2 = A_r2 // Es el array después de las operaciones de los mensajes de sus hijos
   if  p_i es la raíz then
      M_{p_i} = M(↓, (x_modo), max(A_r2))
      return x_modo
   else
      send M_{p_i} = M(↑, A_r1, A_r2) a su padre
   end if
end if
```

Corrección del algoritmo y su complejidad.

Ahora se mostrará que en efecto A_{first} hace lo que tiene que hacer, que en términos formales es decir que A_{first} es correcto, para aquello diremos que es invariante en el siguiente sentido:
Que para cada ronda $r \in \{1, \ldots, h_{r_0}\}$ se cumple que el nodo v de altura h_v ha calculado correctamente su variable local a saber su correspondiente M_v que es un mensaje de la lista de tipos.

Theorem 1. *Sea $\pi = A_{first}$ entonces tiene la propiedad de que para cada $r \in \{1, \ldots, h_{r_0}\}$ al final de la ronda r el nodo v de altura h ha calculado correctamente su variable local M_v y además en el sub-arbol T_v que denota que esta enraizado en v tiene en su variable local las entradas de los nodos de T_v y con posibles repeticiones.*

Demostracion:
Sea $h = 1$ entonces al final de la ronda correspondiente, v es una hoja con base al init del algoritmo $\therefore M_{v_{h=0}} = M(\uparrow, A_{r1}, A_{r2})$

Suponemos que es cierto para el paso $h = j$ y demostremos que es cierto para $h = j+1$ como es cierto para el paso $h = j$ entonces tiene el subarbol T_{v_j} calculada la variable local correspondiente, luego por la construcción de A_{first} se tiene que en v de altura $j+1$ también tiene calculada su variable local que es por la construcción del algoritmo es:
$\therefore v_{h=j+1} = M(\uparrow, A_{r1}, A_{r2})$

Del razonamiento anterior se desprende que el nodo que calcula el máximo, es en efecto la raíz del arbol

Theorem 2. *Para el algoritmo A_{first} en la ronda r el nodo v esta calculando de manera implicita el elemento mas repetido, mas aún en la ronda $r = h_{r_0}$ es el que toma la decisión final de cual es elemento mas repetido y disemina el mensaje a manera de reponse en el resto del arbol*

Demostracion:

Sea r una ronda, entonces notamos que en la correspondiente ronda el vértice r podemos hacer la operación en la variable local M_v que tiene una representación $M(\uparrow, A_{r1}, A_{r2})$ por las lineas de código del algoritmo, en particular se puede calcular $max(A_{r2})$ que por el mapeo implicito entre A_{r1} y A_{r2} le corresponde el elemento mas frecuente en A_{r2} a saber: x_{modo} Y observamos que si $h = h_{r_0}$ y seguimos las lineas de código para ese caso, entonces para $x_{modo_{r_0}}$ es el valor que se toma finalmente y es en la ronda en la que se disemina dicho mensaje.

Ahora se tiene que analizar y deducir la complejidad del algoritmo en el sentido temporal y de mensajes. Por un lado, la complejidad temporal se puede desprender observando la demostración del teorema anterior:
que para la altura correspondiente a la raiz denotada como h_{r_0} será la complejidad temporal pero solo en la parte del algoritmo **request**, pues en el **response** tardara exactamente lo mismo hasta diseminar el mensaje en el resto de los nodos del arbol, entonces se dice más formalmente en el siguiente enunciado:

Theorem 3. *Sea $\pi = A_{first}$ entonces la complejidad temporal denotada como $Time(T_{first}) = O(h_{r_0})$ y la complejidad de mensajes es exactamente $Message(A_{first}) = O(m)$ donde m denota el número de aristas en T_{r_0}*

Demostracion:

*Por construcción del algoritmo y por la corrección del algoritmo, en particular por la propiedad de "Loop-Invariant"podemos observar que se disemina el mensaje hasta el paso de la inducción que es h_{r_0} correspondiente a la altura de de la raíz, y luego en el proceso de diseminar el mensaje en todo el arbol de manera de **response** es la misma altura, entonces la complejidad temporal es $O(h_{r_0})$ Por otro lado, afirmamos que hay tantas rondas como canales de comunicacion, pues en esencia se esta mapeando la cantidad de mensajes con los canales de comunicación, pero eso solo en el proceso de **request** por el diseño del algoritmo, en el proceso de **response** que es la diseminación del mensaje es el mismo mapeo que en el proceso de **request** entonces hay tantas rondas como el doble de canales de comunicacion en el proceso de **request, response**, mas formalmente lo escribimos de la siguiente manera: $Message(A_{first}) = O(m)$ donde m denota el numero de aristas en en el Arbol en el que estamos ejecutando el algoritmo.*

Observacion:

Se observá que existe un subarbol en el que su raíz toma la decision de: **return** x_{modo}
Es decir se dice que se puede optimizar A_{first}, sujeto a la altura h, y apartir de ese subarbol diseminar el mensaje a manera de **response** en todo el arbol T_{r_0}

1.3.2. Decidibilidad

Una vez que se tiene los elementos del modelo distribuido, se introducirá la noción de **decibilidad** en este modelo de cómputo formal.

Definition 25. *Sea w una cadena, se puede escribir a la cadena como $w_0, \ldots, w_n$, la cual será la entrada al algoritmo $\Pi(w)$, donde de manera distribuida, se tendrá como inicialización, que cada proceso p tendrá como entrada un caracter de la cadena w, por decir $p_j(w_k)$.*
Sea $\kappa_{\Pi(w,G)}$ una ejecución del algoritmo Π con entrada w en la gráfica G, entonces se dirá que la entrada w es aceptada, si existe una configuración en la ejecución $\kappa_{\Pi(w,G)}$, por decir C_k tal que existe un estado q_a en C_k, de su correspondiente proceso p_a, tal que $q_a = q_{accept}$.
Es decir, que el estado en esa configuración es exactamente el estado q_{accept}.
En símbolos:

$$\forall \kappa_{\Pi(w,G)},\ \exists C_K \ \mid \exists p_a \ \mid q_{a,k} = q_{accept}. \tag{1.10}$$

Lo anterior, permitirá definir lo siguiente:

Definition 26. *Al conjunto de cadenas que acepta un algoritmo distribuido Π es el lenguaje de Π, o el lenguaje que decide Π, y se denotará como $L(\Pi)$.*

Capítulo 2

Simulación de modelos Maquina de Turing y LOCAL

2.1. Noción de simulación en modelos

Una vez que se tiene estos dos modelos de cómputo formal, a nivel logíco se dice que:

Definition 27. *Se dice que un modelo de computo formal T simula a un modelo de computo formal S si:*

$$\forall x \in L(S) \textit{ entonces } x \in L(T) \tag{2.1}$$

Mas aún se dice que son modelos equivalentes (computacionalmente) si:

$$\forall x \in L(T) \iff x \in L(S) labeleq : equation14 \tag{2.2}$$

Entonces se enunciará el teorema de la siguiente manera:

Theorem 4. *Sea TM una máquina de Turing, entonces existe un Π algoritmo distribuido que simula a TM, con la semántica de* ***simulación****, con base a la definición anterior.*

Una vez que se tiene enunciado este teorema,se dará el paso al diseño del algoritmo distribuido, digamos Π, tal que para toda ejecución $\Theta_{\Pi(w,G)}$ con ambientación el modelo **LOCAL**, con la entrada w, para una gráfica G con una cierta topología es tal que **acepta**.

2.2. Diseño del algoritmo

Remark. *Trivialmente, se pensará que al darle como entrada la cadena que es aceptada por una máquina de Turing* TM*, es consumida tal que cada proceso la tiene enteramente como entrada, i.e* $\Pi(w)$ *entonces de manera local se da que* $p_j(w)$*,* $\forall v_j$*, entonces este proceso en particular es tal que en algún momento de la ejecución(para alguna ejecución* $\Theta_{\Pi(w,G)}$*), exista una configuración* C_k*, y en esta exista un estado* $q_{r,k}$*, tal que* $q_{r,k} = q_{accept}$*, pues su respectivo proceso* p_r *es una máquina de Turing, por lo tanto de manera global,* $w \in L(\Pi)$*, pero se observará que la forma de la entrada en el algoritmo* Π *es de manera no distribuida, pues de manera intuitiva todos los procesos conocen toda la información.*

Entonces no es verdaderamente un diseño de un algoritmo distribuido, por lo tanto se dará paso a diseñar de manera **distribuida** siguiente algoritmo:
Se dará la distribución de la información a manera de contricante de la siguente manera:
Sea $w \in L(TM)$, para una máquina de Turing TM arbitraria, entonces se dicé que una rebanada(pedazo) de la cadena $w[i]$, o con notación de indice w_i, tiene localidad i.
Esta notación y contexto permite definir lo siguiente:

Definition 28. *Sea* p_k *un proceso de una gráfica* G*, con cierta topología, entonces diremos que tendrá una familia* f_k *de posibles entradas, formada por caracteres* w_t*, con* t *localidad para un algoritmo distribuido* Π*, con ambientación en modelo* ***LOCAL***

Esto da la noción del control externo de las entradas, que por el momento esto esta generando una cierta familiaridad del papel a un alto nivel de la visión del contrincante, como se observa esta es la contraparte del algoritmo que esta gobernando computacionalmente (o por la capa de computo) del algoritmo. Entonces se propone el siguiente diseño del algoritmo que dará a priori la solución del problema que estamos atacando.

Algorithm 2 $Simula\_Algo\_TM(w)$

```
for all round ← 1 do
   v_j(w_i){Código para v_j}
   read(w_i)
   while true do
      call δ(q_j, w_i)
      (q_r, w_r, P) ← δ(q_j, w_i)
   end while
   if q_r == q_accept then
      return q_r
   end if
else
   MSG ←< q_r, w_r >
   send(MSG) to Neighbours(j) {vecinos de j}
end for
```

2.3. Descripción del algoritmo

Observando el pseudocódigo del algoritmo, se observa que se hará la iteración por rondas($round$), en virtud del ambiente **local**, luego se hace el código para cada proceso v_k, y la lectura de la entrada w_i, que es la que es por la inicialización o después de una operación $deliver(MSG)$.
Luego se realiza una iteración en la cual se harán llamadas de $\delta()$ y se actualizará la salida de dicho llamado, para repetir este proceso hasta que la localidad de la cadena de salida w_r, sea de localidad no asignada a la familia de símbolos de la cadena, asignada a el proceso actual, el cual es asignado por el lado del contrincate, que es el agente externo del sistema distribuido.
Finalmente se toma la decisión de regresar el estado q_{accept},si el estado $q_r == q_{accept}$; en otro caso se ejecuta la operación $send(MSG)$ a los vecinos del proceso actual v_k.
Lo que sigue, es realizar la demostración que este algoritmo es correcto, lo cual se enunciará en el siguiente teorema.

2.4. Demostración del procedimiento $Simula\_Algo\_TM$

Theorem 5. *El algoritmo Simula_Algo_TM es correcto.*

Demostración. Sea r una ronda de la ejecución del algoritmo $\Pi = Simula\_Algo\_TM$, al incio de esa ronda se estará iniciando un evento del tipo $COMPUTE(k)$, del repertorio de eventos para el proceso v_k, por la naturaleza de la distribución de la información llegará un momento de la iteración en la que se de una estructura de dato del tipo $msg \leftarrow< q_r, w_r, P >$, arrojada por el llamado iterativo de δ, ya que la localidad de w_r no esta asignada a v_k, sin perdida de generalidad.
Entonces, siguiendo el código, se observa que se tiene una lógica para la estructura de dato:
si $q_r == q_{accept}$, entonces $w \in L(\Pi)$, y se da por terminada la ejecución en dicha ronda. Si no, entonces se hace la operación $Send(t, MSG)$, donde sin perdida de generalidad t representa el indice de uno de los vecinos del proceso v_k, por otro lado como $w \in L(TM)$ entonces $\exists v_l$ en la ronda $r + 1$ tal que al final dicha ronda $\exists q_l$ estado tal que $q_l == q_{accept}$.
$\therefore w \in L(\pi)$, por lo tanto el algoritmo es correcto.

□

Una vez que se tiene la corrección del algoritmo, se desprende a manera de corolario la simulación de TM en $LOCAL$.

Corrollary 1. *Sea TM una máquina de Turing, entonces:*

$$\forall w \ \in L(TM) \ \exists \Pi \ algoritmo \ con \ ambientacin \ LOCAL \ t.q \ w \in L(\Pi) \tag{2.3}$$

Demostración. Sean $w \in L(TM)$ para una máquina de Turing y $\Pi = Simula\_Algo\_TM$, $\Pi(w)$ como dicho algoritmo es correcto por el teorema 2, entonces se desprende el hecho de tener un algoritmo en $LOCAL$ tal que $w \in L(w) \ \forall w \in L(TM)$, que en el contexto se reduce a que Π **simula** a TM, con TM una **máquina de Turing** abstracta. □

Entonces una vez que se tiene un algoritmo que es correcto a nivel lógico, la siguiente pregunta es la complejidad asociada a la ejecución de $\Pi \leftarrow Simula\_Algo\_TM$ tanto espacial, de comunicación asi como temporal.

Capítulo 3

Complejidad computacional en los modelos

En esta sección se presentará una de las dimensiones del análisis de algoritmos, a saber la complejidad computacional. Como se esta usando dos modelos de cómputo formal, se darán las definiciones de complejidad en ambos modelos.

3.1. Complejidad computacional en máquinas de Turing

Si se toma un A lenguaje que es decidible, la pregunta que surge es: ¿Cuánto tiempo tardará una máquina de Turing en decidirlo?. Para responder de manera formal y riguroza dicha pregunta, se definirá la complejidad temporal de un algoritmo en este modelo, para ello se pensará en el número de pasos que toma una máquina de Turing para que decida un lenguaje en abstracto, como una función con dominio natural, i.e con dominio el conjunto de los números naturales.

Definition 29. *Sea M una máquina de Turing tal que se detiene para todas sus entradas.*
El tiempo de ejecución o complejidad-temporal asociada a la máquina M es la función $f : \mathbb{N} \to \mathbb{N}$, donde $f(n)$ es el máximo número de pasos que M usa para cualquier entrada de longitud n.

Con lo anterior se dice que si $f(n)$ es el tiempo de ejecución de M entonces M corre en tiempo $f(n)$, y que M es una $f(n)$-máquina de Turing.

Notación Big-oh y small-oh

Con el concepto anterior se darán las siguientes nociones de teoría de funciones, las cuales darán una notación compacta para dar la medida de la complejidad o tiempo de ejecución de una máquina M con entrada I. Esto se hará con el fin de dar una aproximación al tiempo de ejecución de una máquina M, dada la compljidad de la misma.
Con eso en mente damos paso a definir lo siguiente:

Definition 30. *Sean $f, g : \mathbb{N} \to \mathbb{R}^+$ funciones*
Se dice que $f(n) = O(g(n))$

si $\exists c, n_0$ *tal que* $\forall n >= n_0$

$$f(n) <= cg(n) \tag{3.1}$$

Cuando $f(n) = O(g(n))$*, se dice que* $g(n)$ *es un limite asintoticamente superior para la función* $f(n)$*.*

Esta definición formaliza la noción de medir el tiempo de ejecución para una máquina de Turing M, en términos de aproximar dicha función, ya que intuitivamente $f(n) = O(g(n))$ dice que f es menor o igual que g, si se despreciamos las diferencias hasta un factor constante.

A continuación se dará un ejemplo de una máquina de Turing en la que se hará analísis de la complejidad temporal de la misma.

ejemplos

Sea M una máquina de Turing, tal que hace decidible al lenguaje:

$$A = \{0^k 1^k \| k \in \mathbb{N}\} \tag{3.2}$$

Procedamos a describir la máquina como procedimiento:

1. Escanear a través de la cinta y rechazar si un 0 es encontrado a la derecha de un 1.
2. Repetir si tanto 0s como 1s permanecen en la cinta.
3. Escanear a través de la cinta, tachando un simple cero y un simple 1
4. Si todavía quedan 0s después de haber tachado todos los 0s o si aún quedan 1s despues de haber tachado todos los 0, **reject**
5. En otro caso si no quedan ni 0s ni 1s entonces **accept**

Se sigue el análisis de esta máquina de Turing, para ello, se hará la siguiente observación:
Se considerará los estados de esta máquina por separado. En el estado 1, la máquina escanea a través de la cinta para verificar que la entrada es de la forma 0*1*. Al ejecutarse dicho escaneo en n-pasos, donde n es para representar la longitud de la cadena como entrada. Reposicionando el cabezal en el final del lado izquierdo de la cinta, nuevamente en n-pasos. Entonces el total usado en este estado es de $2n$-pasos. Luego en notación **big-O**, se dice que este estado usa $O(2n)$ pasos, que por la deficion anteror esto es asintoticamente equivalente a $O(n)$-pasos.

Luego en los estados 2 y 3, se sigue que la máquina de manera repetitiva escanea la cinta que en terminos de las operaciones por las que ha sido dotada por definicion este objeto de cómputo es una operación de lectura, y ademas tacha(escribe) con una x a los 1 y a los 0 en cada lectura de la cinta. En cada escaneo(lectura) usa $O(n)$-pasos, ya que en cada escaneo tacha(escribe) dos símbolo, a lo más en $n/2$ escaneos(lectura).

Por lo tanto el tiempo consumido en los estados 2 y 3 es $(n/2)O(n)$ y como el factor no es una constante entonces eso es igual a $(1/2)O(n^2) - pasos$ que eso es asintóticamente equivalente a $O(n^2) - pasos$. Y finalmente en el estado 4, la máquina simplemete toma la decisión de aceptar o

rechazar, luego el tiempo tomado en dicho paso es $O(n)$.
Luego en conclusion el tiempo tomado de M_1 con entrada de longitud n es

$$O(n) + O(n^2) + O(n) = O(n^2 + 2n) = 0(n^2) \tag{3.3}$$

Luego por definición se sigue que la complejidad temporal o tiempo de ejecución es de $O(n^2)$, lo que termina el análisis temporal de M.

Se definirá un concepto que hara una segementación del conjunto de máquinas de Turing por la complejidad temporal asociada a ellas.

Definition 31. *Sea $t :\to R^+$ una función.*
*Se define la **clase** del tiempo de complejidad*

$$TIME(t(n)) \tag{3.4}$$

Como la colección de todos los lenguajes tal que son decidibles para una máquina de Turing en tiempo $O(t(n))$.

Por lo tanto, con base a la definición anterior, el lenguaje $A\{0^k 1^k \| k >= 0$ es de clase $TIME(n^2)$, ya que la complejidad temporal asociada es $O(n^2)$.
Se puede observar que aunque exista una máquina de Turing que decida a un lenguaje en tiempo $t(g(n))$ con $g :\to^+$ función, puede existir otra máquina que la decida asintóticamente más rápido que la anterior i.e

$$TIME(t(n)) \; para \; t(n) = o(g(n)) \tag{3.5}$$

con $t : \mathbb{N} \to \mathbb{R}^+$

3.2. Complejidad computacional en modelo distribuido $LOCAL$

3.2.1. Complejidad temporal

El tiempo de complejidad de un algoritmo secuencial es medido como el número de pasos que toma desde que se comienza hasta que termina. Pero ello no es suficiente en ambientes distribuidos, es decir no basta con contar el numero de pasos que termina la ejecución del mismo, como por ejemplo en un ambiente computacional multiprocesos, existe la posibilidad que cuando un programa segmenta el algoritmo involucrado en varios procesos tal que son intercambiados dentro y fuera de la ejecución en diferentes tiempos, resultando una situación donde un proceso por un lado espera mensajes de otro proceso, tal que este esta actualmente detenido.

Para los propositos de este analísis, asumiremos que el algoritmo esta ejecutado en este sistema de manera independiente, sin tener otras actividades que tienen lugar concurrentemente. Sin embargo, los retrasos ocurren en cada proceso como resultado de tener que esperar información computada de otro proceso, los cuales no pueden ser ignorados.

Como podemos ver no es suficiente con contar el número de pasos que toma cada proceso, también tenemos que agregar el número de huecos que toma en su funcionamiento en dicho conteo.

Formalmente la complejidad temporal se define como sigue:

Definition 32 (Complejidad temporal síncrona:). *El tiempo de complejidad o tiempo de ejecución de un algoritmo distribuido* Π *en una red* G*, denotado como* $TIME(\Pi, G)$*, es el número de pulsos generados durante la ejecución de* Π *en* G*, en el peor caso, desde que el primer proceso inicia la ejecución hasta que el último hallá teminado. Todo esto con una entrada legal* I *en* G *y en un escenario de ejecución.*

Remark. *Existe una definición para el mundo de algoritmos asíncronos, pero para los fines de esta tesis nos limitaremos a esta definició.*

3.2.2. Complejidad espacial.

Ahora vamos a definir la complejidad espacial o complejidad de memoria, para ello se tomará en consideración la memoria requerida para el algoritmo, y para ello se tomará en alunos casos la máxima memoria local requrida en cualquier vértice.

Definition 33 (Complejidad de memoria:). *El total de complejidad espacial de un algoritmo* Π *en una red* G*, denotada como* $MEM(\Pi, G)$*, es definido como el número total de bits usados por el algoritmo en la red, en el peor de los casos. El máximo espacio de complejidad de un algoritmo* Π *en la red* G *denotado como* $MAX\_MEM(\Pi, G)$*, es el máximo número de bits usados por el algoritmo en cualquier proceso de la red, en el peor de los casos. Todo esto con una entrada* I *legal en* G*, y cualquier escenario de ejecución.*

Estas do etiquetas($MEM(\Pi, G), MAX\_MEM(\Pi, G)$) son para medir la complejidad de memoria, en particular $MAX\_MEM(\Pi, G)$ permitirá identificar algoritmos logrando una distribución de memoria mas equilibrada con base a los requerimientos de la misma.

3.2.3. Complejidad de mensajes

En este modelo distribuido se agregará una tercera capa de complejidad, que medirá el costo de comunicación en la red. Para ello observemos que en un modelo centralizado se tiene por el lado de la complejidad temporal el siguiente contexto:

1. Plazo de finalizacion, nombrado, el tiempo en el que el cliente(s) puede esperar obtener el resultado de su cómputo, y
2. el costo, a saber, el número esperado de las operaciones requeridas para su cómputo.

Estos dos factores, en una ambientación distribuida no estan relacionados mutuamente. Pues la computación es por la distribución del trabajo entre los proceso. Luego la estimación del plazo de final es aún lograda por la noción de complejidad temporal, el costo de la computación es ahora evaluada usando la noción de complejidad de mensajes, como la medida principal.
De hecho, la complejidad de mensajes es la de mayor costo en la ejecución de un algoritmo distribuido.
La definición de complejidad de mensajes, será definida de la siguiente manera.
La longitud basica de un mensaje MSG, se asume que es $O(log(n))$ bits.

Definition 34 (Complejidad de mensajes:)**.** *El costo de mensajes de transmitir un mensaje basico sobre un canal de comunicación tiene por valor 1. La complejidad de mensajes de un algoritmo distribuido* Π *en una red* G*, denotado como* (Π, G)*, es el número total de mensajes basicos transmitidos durante la ejecución de* Π *en* G *en el peor de los casos. Todo bajo en una entrada legal* I*, y en cualquier escenario de ejecución.*

3.2.4. Complejidad del algoritmo