UNIVERSIDAD DEL PAPALOAPAN

Campus Loma Bonita

LICENCIATURA EN MATEMÁTICAS APLICADAS

ANÁLISIS DE LA COMPLEJIDAD EN ALGORITMOS DE CALENDARIZACIÓN

TESIS

QUE PARA OBTENER EL TÍTULO DE :

LICENCIADO EN MATEMÁTICAS APLICADAS

PRESENTA :

ALEXIS MENDOZA MOGUEL

DIRECTOR DE TESIS :

DR. JUAN MANUEL PÉREZ ABARCA

LOMA BONITA, OAXACA. 2018

UNIVERSIDAD DEL PAPALOAPAN

Campus Loma Bonita

LICENCIATURA EN MATEMÁTICAS APLICADAS

LA PRESENTE TESIS TITULADA **"ANÁLISIS DE LA COMPLEJIDAD EN ALGORITMOS DE CALENDARIZACIÓN"** PRESENTADA POR EL SUSTENTANTE DE LICENCIATURA **ALEXIS MENDOZA MOGUEL** BAJO LA DIRECCIÓN DEL **DR. JUAN MANUEL PÉREZ ABARCA**, HA SIDO REVISADA Y ACEPTADA POR EL COMITÉ EXAMINADOR PARA SER DEFENDIDA EN EL EXAMEN PROFESIONAL Y OBTENER EL TÍTULO DE LICENCIADO EN MATEMÁTICAS APLICADAS.

DR. JUAN MANUEL PÉREZ ABARCA
DIRECTOR

DRA. BEATRIZ CARELY LUNA OLIVERA
PRESIDENTE

DR. EDUARDO SÁNCHEZ SOTO
SECRETARIO

DR. JUAN MANUEL PÉREZ ABARCA
VOCAL

LOMA BONITA, OAXACA, MÉXICO. 2018

Dedicado a mi familia

Agradecimientos

El presente trabajo es el resultado de múltiples decisiones tomadas a lo largo de mi estancia en la universidad, en especial de aquellas que me llevaron a conocer gente extraordinaria que aportó grandes experiencias, ideas, pasión y también virtudes que hoy son parte de mi personalidad, mi más sincero agradecimiento para esas personas.

Debo dedicar éste trabajo a mi familia especialmente a mi madre Antonia, a mi tía Anita y mi querido hermano Leví que son el pilar fundamental de mi vida y que gracias a su paciencia, dedicación y amor han hecho posible la realización de este logro en mi vida. También debo agradecer a mi tía Refugio, a mi tío Crescencio y a primo Emmanuel por su invaluable ayuda en diversas ocasiones.

Agradezco al Dr. Juan Manuel Pérez Abarca mi director de tesis, el haber compartido sus experiencias, su conocimiento y por todos los consejos que hicieron posible la comprensión del tema que aquí se expone, le estoy eternamente agradecido.

Le doy las gracias a todos mis profesores, muy en especial al M. C. Xaap Nop Vargas Vázquez, a la Dra. Beatriz Carely Luna Olivera, al Dr. Eduardo Sánchez Soto, al Dr. José Geiser Villavicencio Pulido, a la M. C. Claudia Nila Luevano y al Dr. Marcelino Ramírez Ibáñez por todos los consejos y palabras de ánimo que fueron pieza clave en los momentos más difíciles. A mis amigos Cristobal, Maricela, Félix, Cesar y Amayrani por todos esos grandes momentos y por todas las aventuras que pasamos juntos.

Quiero dar las gracias a la Universidad del Papaloapan por todo el apoyo brindado a mi persona, porque el esfuerzo de cada una de las personas que laboran en ella hace posible la educación de alto nivel en mi región.

Finalmente quiero dar las gracias al pueblo de México, a campesinos, maestros, obreros y a todos aquellos mexicanos que se levantan todos los días a realizar sus jornadas de trabajo para hacer de éste un mejor país, ya que con sus contribuciones hacen posible la educación publica y de calidad, y yo no estaría escribiendo estas palabras de no ser por ello, mil gracias a todos y todas.

En memoria de: S. M. R. y P. M. C.

Resumen

El desarrollo de cualquier actividad productiva o de servicio se requiere de la organización adecuada de las tareas y recursos que se disponen para realizar dicha labor con éxito. La teoría de la calendarización es la disciplina científica que surgió a partir de esta problemática y uno de sus principales exponentes fue Henry Laurence Gantt. En la presente investigación se estudian los temas referentes a las técnicas usadas en los modelos de máquinas secuenciales y su relación con la complejidad computacional, como se verá en el desarrollo del presente trabajo, la gran mayoría de los problemas de calendarización de una máquina pertenecen a la clase **NP**-díficil y algunos parecen ser inclusive **NP**-completos, dichos conceptos no son sencillos de formular de una manera simple, por lo que fué necesario realizar un estudio relativamente amplio sobre la computación clásica con la finalidad de establecer con precisión la naturaleza de la dificultad de los algoritmos de calendarización.

Aunque existe diversas técnicas para resolver problemas de calendarización, nos restringiremos a estudiar el método de reducción y algunas técnicas de algoritmos generales haciendo caso omiso del procedimiento de ramificación y acotamiento (branch and bound), programación dinámica, entre otras.

Palabras clave : Calendarización, complejidad, reducción.

Abstract

The development of any productive or service activity requires the adequate organization of the tasks and resources that are available to carry out this task with success. The theory of scheduling is the scientific discipline that emerged from this problem and one of its main contributors was Henry Laurence Gantt. In the present investigation the subjects related to the available techniques in the models of machines and their relation with the computational complexity are studied, as it will be seen in the development of the this work, most of the scheduling problems of one machine belong to the class **NP**-hard and some of them even seem to be **NP**-complete, such concepts are not easy to formulate in a simple way, therefore, a quite broad study about classical computing is presented in order to establish with precision the nature of the difficulty of the scheduling algorithms.

Although there are several techniques to solve the scheduling problems , we will restrict ourselves to study the method of Reduction and some general algorithms ignoring the procedure of branch and bound and dynamic programming, among others.

Keywords: Scheduling, complexity, reduction.

Contenido

Índice de figuras

Índice de tablas

Introducción

En casi todos los dominios de cualquier empresa la asignación de recursos limitados en un período de tiempo, por ejemplo: personal, máquinas, herramientas etc, para el procesamiento de pedidos (trabajos) dentro de un cierto tiempo, es en general una actividad imprescindible, crítica y difícil. Los retos que se presentan en éste proceso son parecidos y se pueden plantear y resolver de manera general, independientemente de si se trata de problemas de logística, producción, suministro, mercadotecnia o inclusive de acceso a bases de datos y diseño de sistemas operativos. Incluso, aunque no se pueda ver a simple vista, es extraordinariamente similar la asignación de pedidos de producción a diferentes máquinas y la cuestión de qué aplicación se debe procesar en cual núcleo de un multiprocesador.

La lista de tareas posibles cuya realización requiere la asignación de ciertos recursos parece interminable, por mencionar algunos diremos: el movimiento de aviones requiere de pistas de aterrizaje y despegue, los pedidos de clientes se procesan por un oficinista, miembros de una familia que se deben duchar antes del desayuno etc.

Es común a todas estas tareas, el hecho de que los recursos necesarios solo están disponibles en cantidad limitada. De esta manera, nos acercaremos al tema, considerando la planeación o calendarización de una forma muy general. Así que haremos esencialmente tres suposiciones:

1. Cada recurso solo se puede asignar en un momento dado a una sola tarea.
2. Los recursos no son consumibles, siempre estan a disposición una vez liberados.
3. Una tarea solo ocupa un recurso en un momento dado.

La segunda suposición deja fuera a los problemas relativos a materiales que se consumen por ejemplo gasolina o dinero.

En lo sucesivo utilizaremos el término “máquina” en lugar de “recurso”, también utilizaremos indistintamente el término “pedido”, “tarea” u “órden” como sinónimo de trabajo. Igualmente, nos referiremos al problema de asignación de recursos, ya sea como planeación, plan, calendario, calendarización o incluso programa.

Capítulo 1

Elementos de la calendarización

"El tiempo es el recurso más importante; quien no lo sabe administrar no sabe administrar nada."

Peter Drucker.

1.1. Notación y ejemplos

El objetivo de este capítulo es introducir mediante ejemplos clásicos la notación tradicionalmente aceptada para representar las particularidades de un problema dado de calendarización, así como ilustrar las técnicas básicas de solución cuando estas existen y finalmente hacer ver las dificultades de problemas más amplios.

En la situación, en que n pedidos se deben procesar en una máquina específica, para cada pedido j, $j \in \{1,\ldots,n\}$, denotaremos por:

p_j: Tiempo de procesamiento.

d_j : Tiempo de entrega comprometida de la tarea.

Para un calendario establecido, sea C_j el momento en que la tarea j se ha terminado de procesar, entonces se define el retardo L_j de la tarea como: $L_j = C_j - d_j$, es importante entonces responder a la pregunta: ¿en qué orden se deben procesar los trabajos? de manera que se minimice el retardo máximo:

$$\max_{j\in\{1,\ldots,n\}} \{L_j\}$$

Es decir, se busca una permutación $S : \{1,\ldots,n\} \to \{1,\ldots,n\}$ (el calendario de trabajo), el cual específica el orden de las tareas. $S(j) = k$ significa entonces que la tarea j se debe procesar en la posición k.

Veamos un ejemplo sencillo, consideremos $n = 5$ tareas, con los tiempos de proceso y entrega mostrados en la siguiente tabla:

j	1	2	3	4	5
p_j	7	8	10	6	4
d_j	2	14	6	8	18

Dado el criterio de optimalidad establecido, se verá en seguida que el calendario $S(j) = j$, ésto es, la calendarización que ejecuta a cada tarea de acuerdo a sus tiempos de entrega ordenados en forma creciente, en la figura siguiente se ilustra una posible solución:

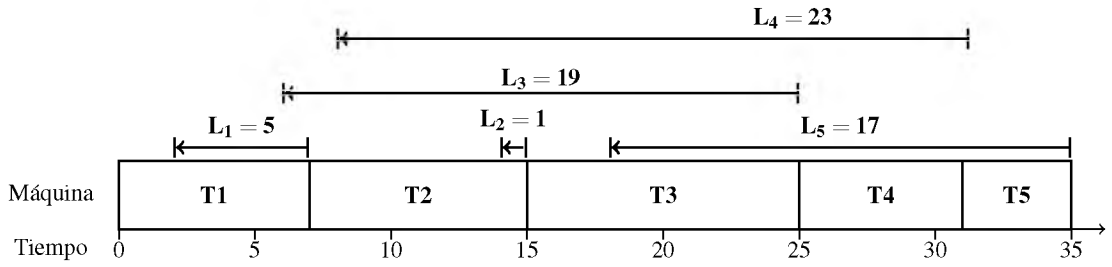


Figura 1.1: Calendario con S(j)=j.

En ella, la punta de las flechas indica el tiempo de entrega comprometido y encima de ellas se escribe el retraso L_j, el resto es autoexplicativo. En dicho calendario, se tiene que $L_{max} = 23$.

La solución propuesta es intuitivamente correcta, pero aún así daremos su demostración, pues nos servirá de referencia para otros problemas de mayor complejidad y lo llamaremos por comodidad la **regla del tiempo de entrega**.

Teorema 1.1. *La regla del tiempo de entrega proporciona el mínimo de*

$$\max_{j\in\{1,\ldots,n\}} \{L_j\}.$$

Demostración. Veremos que cualquier calendario S' que minimiza la cantidad anterior debe ser igual al calendario $S(j) = j$. Podemos también suponer que los pedidos están hechos de manera que $d_1 \leq d_2 \leq \cdots \leq d_n$ y sea $k := \max\{S(j) \mid j \in \{1,\ldots,n\}, S(j) < j\}$, es decir, k es la posición de la última de tareas que se programan antes de su lugar de acuerdo a la regla del tiempo de entrega, claramente se cumple $S(j) = j, \forall j > k$ es decir, el plan se ajusta a la regla del tiempo de entrega a partir del trabajo $k+1$ (si es que existe). Consideremos ahora el calendario S' que se obtiene de la siguiente manera:

$$S'(j) := \begin{cases} j & j \geq k \\ S(j) - 1 & S(k) < S(j) \leq k \\ S(j) & S(j) < S(k) \end{cases}$$

Es decir, el calendario S' regresa el pedido a la posición que debería estar de acuerdo a la regla del tiempo de entrega del pedido k (primera linea), los trabajos comprendidos entre el primero y el $k-1$ (tercera línea) se dejan en su lugar y los restantes se recorren una posición (segunda línea), esto se ilustra en seguida para una situación en que $n = 8$.

j	1	2	3	4	5	6	7	8
S	6	7	2	1	3	4	5	8
S'	6	2	1	3	4	5	7	8

Para los valores anteriores se tiene $\{S(j) \mid j \in \{1,\ldots,n\}, S(j) < j\} = \{6,7\}$ y $k = 7$, obsérvese además que en este caso $S(7) = 2$ y $S'(7) = 7$.

Entonces, la función objetivo de S' no puede ser mayor que la de S, puesto que a lo más se aumenta el retardo a partir del pedido k, el cual no puede ser mayor que el retraso del pedido j en la posición $k = S(j)$ en el plan S. Entonces se tiene que para el plan S' se cumple $S'(j) = j, \forall j \geq k$ es decir el plan satisface la regla de tiempo de entrega a partir de la del pedido k. ∎

Realizando esta operación cuantas veces sea necesario, se debe obtener el calendario que cumple la regla del tiempo de entrega.

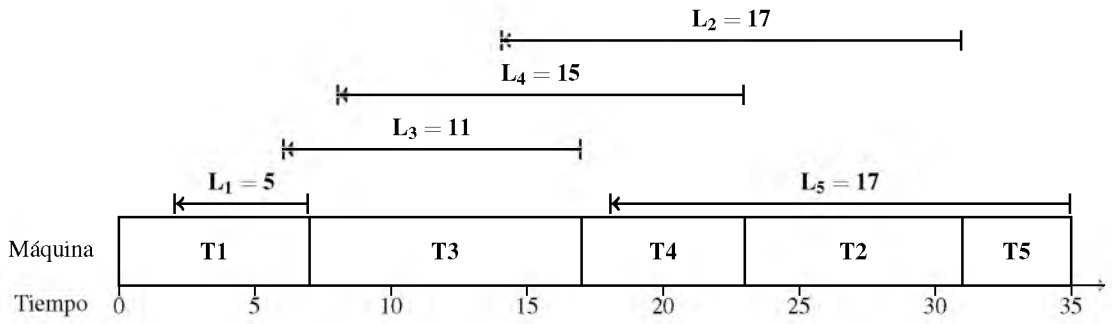


Figura 1.2: Calendario óptimo.

Consideremos ahora una ligera variante del problema anterior, en ésta se trata de minimizar el número de trabajos retrasados, donde el trabajo j esta retrasado si se cumple $C_j > d_j$ obviamente. Entonces se trata de minimizar $|\{j \in \{1,\ldots,n\} \mid C_j > d_j\}|$.

Para ilustrar la dificultad, tomemos el siguiente ejemplo:

j	1	2	3	4	5
p_j	8	4	4	8	6
d_j	9	10	12	14	16

Si tomamos inicialmente el calendario $S(j) = j$, se puede ver en la figura 1.3 que el número de trabajos atrasados es cuatro, de donde surge la pregunta de si hay una secuencia de calendarización óptima.

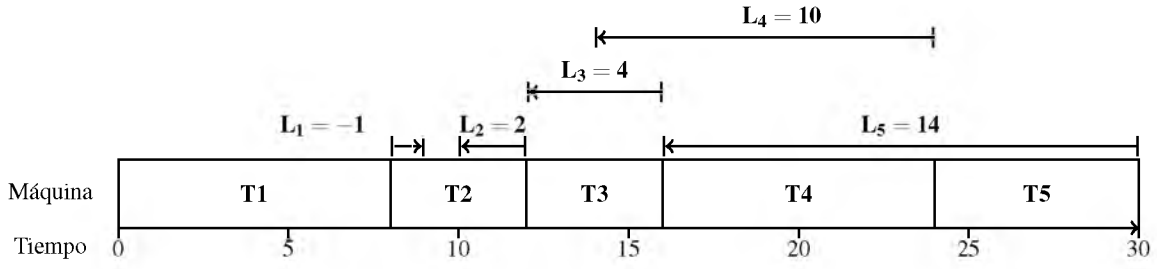


Figura 1.3: Calendario S(j)=j.

También éste problema se puede resolver fácilmente, pero de una manera diferente a la anterior. El principio de base es el siguiente: para la función objetivo es indiferente la magnitud del retardo, solo le interesa saber si un trabajo esta retrasado ó no, de manera que en cuanto sepamos que un trabajo está irremediablemente retrasado, entonces se deja para ser planificado hasta el final. El procedimiento de Moore se basa en esta idea [13].

A partir de una planeación de acuerdo con la regla del tiempo de entrega, se prueba si un trabajo se retrasa, entonces se le pone al final del plan y se aumenta en uno el número de trabajos retrasados al cual denotaremos como U. Entonces tenemos el siguiente algoritmo:

Algoritmo 1 (Procedimiento de Moore)

1. Inicialización: Ordenar los trabajos en forma creciente de acuerdo a su tiempo de entrega, podemos suponer que: $d_1 \leq d_2 \cdots \leq d_n$, entonces el calendario inicial es $S(j) = j, \forall j \in \{1,\ldots,n\}$ y se hace $U := 0$.

2. Criterio de fin: Si $C_j \leq d_j, \forall j \in \{1,\ldots,n\}$ que satisface $S(j) \leq n - U$, terminar.

3. Determinación del trabajo que se va a recorrer: Sea $k := argmin\{j \mid C_j > d_j, S(j) \leq n - U\}$ el trabajo que primero se programa dentro de todos los que están retrasados.

Sea $l := argmax\{p_j \mid 1 \leq S(j) \leq S(k)\}$ el trabajo con el mayor tiempo de procesamiento entre los primeros $S(k)$ pedidos.

4. Recorrimiento del trabajo: Defínase el siguiente calendario:

$$S'(j) := \begin{cases} n & \text{Si} \quad j = l, \\ S(j) - 1 & \text{Si} \quad S(l) < S(j) \leq n, \\ S(j) & \text{Otro caso.} \end{cases}$$

Hacer $S := S', U := U + 1$ e ir al paso 2.

En el segundo paso del algoritmo solamente se consideran los pedidos que todavía no se han recorrido hasta el final. Si ya no hay pedidos pendientes se termina el algoritmo. De otra manera en el paso tres se determina el pedido k que es el primero de los pedidos retrasados. Entre los primeros $S(k)$ pedidos habrá con seguridad justamente uno, el cual se puede recorrer hasta el final. Entonces lo más adecuado es elegir el proceso más largo. En el paso 4 se recorre dicho proceso hasta el final y todos los pedidos entre la posición vieja y la nueva se recorren una posición hacia adelante.

Si ejecutamos el algoritmo al ejemplo dado, el primer pedido retrasado es el 2 como se puede ver en la figura siguiente:

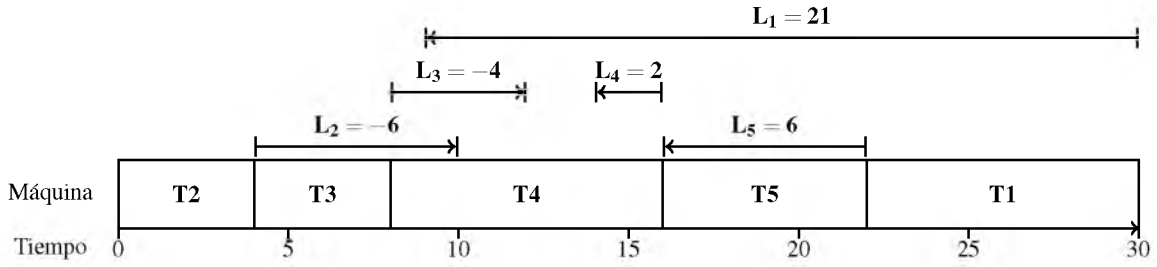


Figura 1.4: Calendario recorriendo el primer trabajo hasta el final.

Por lo que con seguridad, en cualquier calendario uno de los dos trabajos, ya sea el primero o el segundo estará retrasado y como para el algoritmo es indiferente que tan grande es el retraso, se escoge entonces el pedido 1 por ser más largo que el pedido 2 y se le recorre hasta el final, véase la figura 1.4. En seguida el contador de trabajos retrasados U con seguridad se hace igual a 1. Por cierto, entre los primeros $n-U=4$ se encuentra con seguridad uno que está retrasado (el pedido 4) y puesto que se trata del más largo, se le recorre hasta el final. U se hace igual a 2 y como entre los primeros $n-U=3$ trabajos ya no hay ninguno retrasado, éste debe ser el calendario óptimo. De manera que la función objetivo es 2.

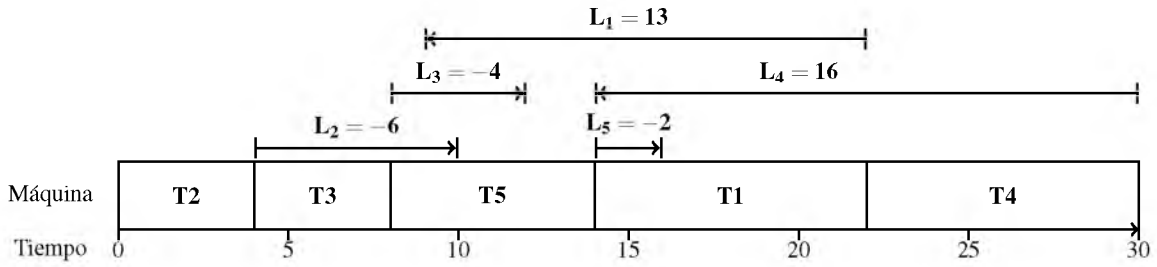


Figura 1.5: Calendario óptimo.

La representación gráfica que hemos utilizado para visualizar el procesamiento de los trabajos es una técnica estándar y se conoce como diagramas de Gantt.

1.2. Características de las máquinas, Propiedades del calendario y Función objetivo

En la sección anterior se introdujeron los conceptos básicos y notaciones del problema de calendarización, también se hizo evidente que cada problema específico está determinado por las características de la máquina, las propiedades particulares del calendario y finalmente de la función objetivo.

La siguiente tabla muestra una lista de las notaciones más usuales, algunas ya han sido utilizadas en los ejemplos anteriores.

n	Número de trabajos o pedidos.
m	Número de máquinas (recursos).
i	Índice de la máquina.
j	Índice del pedido.
S_i	Calendario de la máquina i Permutación del plan $\{1,\ldots,n\}$.
$S_i(j)$	Posición en el calendario del pedido j en la máquina i.
p_j	Tiempo de proceso del pedido j.
p_{ij}	Tiempo de proceso del pedido j en la máquina i.
r_j	Tiempo de llegada del proceso j, es decir el momento más temprano en el que el pedido se puede planear (release date).
d_j	Tiempo de entrega del trabajo j, momento en que el trabajo j debería estar terminado (due date).
w_j	Importancia del pedido j, por ejemplo valor, costo, prioridad (peso).
C_{ij}	Tiempo en el que termina el pedido j al ser procesado en la máquina i.
C_j	Tiempo de terminación del pedido j.

Tabla 1.1: Notaciones usuales en teoría de la calendarización.

Definición 1.1. *Se entiende como un* ***problema de calendarización*** *al problema de optimización representado por la tripleta $\alpha|\beta|\gamma$, en el que se asignan recursos a pedidos (con las propiedades anteriores) y se precisan las propiedades de los recursos (α), las propiedades del calendario y condiciones de frontera (β) así como la función objetivo que se va a minimizar (γ).*

Esta representación de los problemas de calendarización se debe a Graham et al. (1979), quienes la introdujeron y se conoce como la notación de tres campos [24]. A continuación se presentan las propiedades de las máquinas más comunes, las cuales se especifican en el campo α.

1 *Solo hay una máquina, el caso* $\alpha = 1$ *será el objetivo de nuestro trabajo.*

Pm: *Hay m máquinas idénticas que funcionan en paralelo. Un pedido se puede ejecutar solo en una de las máquinas.*

Fm: *En un* flujo de taller *también hay m, pero cada trabajo debe pasar por las m máquinas. El orden de las máquinas en que se debe ejecutar cada pedido esta predeterminado y es el mismo para todos los pedidos.*

Jm: *Un piso de trabajo* Job Shop *es una generalización del flujo de trabajo. Cada trabajo debe pasar solo por algunas máquinas (ó todas) pero el orden puede ser diferente para cada pedido.*

Om: *Taller abierto (*Open Shop*), aquí, todo trabajo debe procesarse en cada una de las máquinas, pero el orden es arbitrario.*

En el caso del campo β*, éste puede tomar uno, varios o ninguno de los siguientes valores:*

$\mathbf{p_j = p}$: *Es el caso especial en que cada trabajo tiene el mismo tiempo de procesamiento.*

$\mathbf{d_j = d}$: *Caso en que todos los trabajos tienen el mismo tiempo de entrega.*

$\mathbf{r_j}$: *Denota que hay un tiempo determinado a partir del cual el proceso se puede procesar* (release date) *y que debe considerarse en la calendarización.*

prmtn: *Si está presente significa que cualquier trabajo se puede interrumpir durante su ejecución, dicho de otra forma, si este campo no esta presente, significa que todos los trabajos se deben procesar de principio a fin sin interrupción.*

prec: *Esta entrada se pone cuando hay una secuencia de precedencia entre los trabajos, la cual se denota normalmente como* $i \rightarrow j$ *para indicar que el trabajo i debe ejecutarse antes del trabajo j.*

$\mathbf{s_{jk}}$: *Con este campo se específica que hay un tiempo de preparación de las máquinas que depende del orden de los trabajos.*

El campo β *indica la función objetivo, a continuación se da una lista de las más usuales, las cuales se entiende que deben minimizarse en la calendarización:*

$\mathbf{C_{max}}$: *La duración total del calendario (makespan), es el momento en que se han terminado todas las tareas, también se puede representar como* $C_{max} := max\{C_1, \ldots, C_n\}$.

$\mathbf{L_{max}}$: *Como se mencionó en la discusión anterior, el retraso de un trabajo esta dado por* $L_j = C_j - d_j$. *por lo que el retraso máximo es* $L_{max} := max\{L_1, \ldots, L_n\}$, *notese que el retraso puede ser negativo cuando el pedido se termina antes de su tiempo de entrega.*

ΣC_j : *Es la suma de los tiempos de terminación, este criterio se adapta a situaciones en las que se debe terminar los costos intermedios de almacenamiento. La minimizacion de ΣC_j es una idea también del valor medio de los tiempos de terminación dado por $\Sigma C_j/n$.*

$\Sigma w_j C_j$: *Es una generalización de la anterior, como se mencionó anteriormente, es posible asignar prioridades a los trabajos mediante el* peso w_j.

ΣT_j : *Se define el* exceso *del pedido j como el valor $T_j := max\{C_j - d_j, 0\} = max\{L - j, 0\}$ el cual solo considera retrasos positivos.*

$\Sigma w_j T_j$: *Es una generalización de la función objetivo anterior.*

ΣU_j : *Es el número de trabajos retrasados, el cual se define por medio de la función:*

$$U_j := \begin{cases} 1 & Si \quad D_j > d_j, \\ 0 & Otro\ caso. \end{cases}$$

$\Sigma w_j U_j$: *Cuyo significado es obvio, en el, se trata de favorecer los trabajos de cierta clase.*

Es claro que no se tienen que considerar todos los campos anteriores en un problema de calendarización, los mencionados aquí son algunos entre una gran cantidad de campos con distintas características que se pueden encontrar en la literatura.

La diferencia entre las diferentes funciones objetivo y su dependencia del tiempo de entrega se ilustra en la siguiente figura. Como se puede ver el retraso L_j es lineal y considera una recompensa en el caso de terminar antes del tiempo de entrega, en cambio en T_j no existe la recompensa, en la función U_j solo es importante si hay o no retraso, finalmente el valor real C_j es en general no lineal.

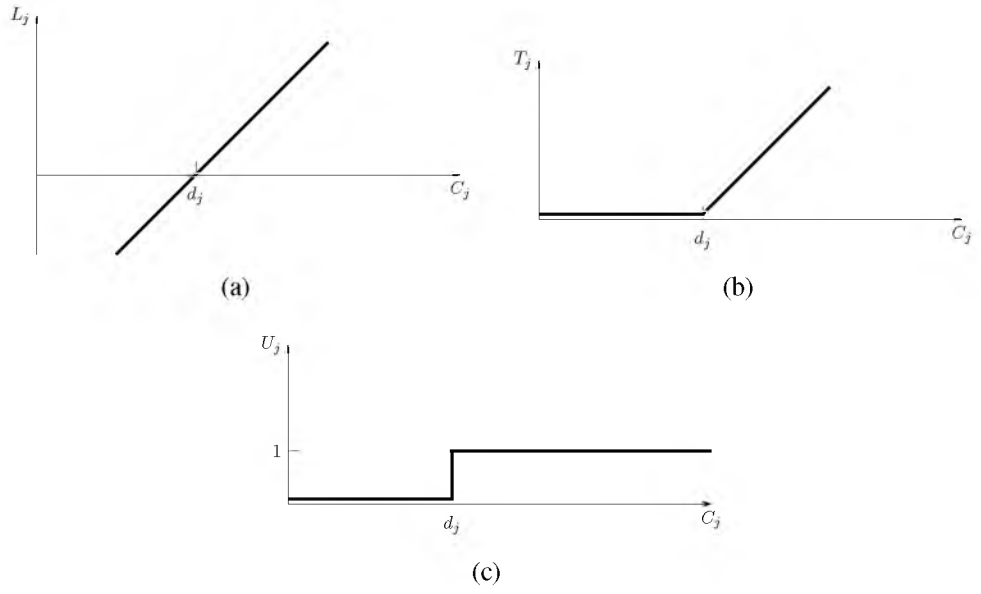


Figura 1.6: Funciones objetivo más usuales.

Consideremos ahora otro problema de calendarización "fácil" de resolver.

Ejemplo 1.1. *Consideremos el siguiente problema de calendarización de la clase* $(1||\Sigma w_j C_j)$*:*

j	1	2	3	4
p_j	3	5	6	8
w_j	5	10	9	1

Antes de resolverlo, es importante preguntarse ¿hay algún trabajo que deba planearse antes o después de los otros?, ¿hay algún trabajo que se tenga que programar necesariamente antes o después de otro? El calendario óptimo es $S = (2,1,3,4)$ *del cual se calcula fácilmente su función objetivo como 238, uno puede también verificar que si ordenamos los trabajos por orden creciente del valor* p_j/w_j*, entones se obtiene el calendario óptimo.*

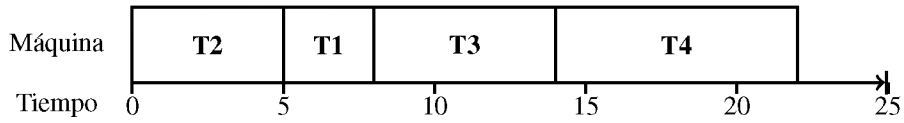


Figura 1.7: Calendario óptimo.

A esta regla se le conoce como la regla del tiempo de procesamiento promediado más corto (WSPT por sus iniciales en inglés). En el siguiente teorema veremos que este ordenamiento conduce al óptimo buscado. □

Teorema 1.2. *La regla WSPT proporciona el valor óptimo del problema* $(1||\Sigma w_j C_j)$*, es decir que si* S *es una solución óptima del problema para el cual se cumple* $S(j) < S(k)$ *entonces se debe cumplir* $w_j/p_j \geq w_k/p_k$*.*

Demostración. Supongamos que en el calendario óptimo S hay dos pedidos j y k con $S(j) < S(k)$ pero se tiene $w_j/p_j < w_k/p_k$, podemos suponer que estos pedidos son contiguos, es decir $S(j)+1 = S(k)$, de otra manera existiría un pedido l tal que $S(j) < S(l) < S(k)$ y además $\frac{w_k}{p_k} > \frac{w_j}{p_j} \geq \frac{w_l}{p_l} \geq \frac{w_k}{p_k}$ lo cual es una contradicción.

Sea c el momento de inicio de j en S. Con esto, los pedidos j y k contribuyen a la función objetivo de s con la cantidad: $(c+p_j)w_j + (c+p_j+p_k)w_k = cw_j + p_j w_j + cw_k + p_j w_k + p_k w_k$. Si ahora consideramos el calendario S' que se obtiene de S solamente intercambiando los pedidos j y k, es decir $S'(j) = S(k)$ y $S'(k) = S(l)$, para el se cumple $S'(l) = S(l)$ para toda $l \neq j,k$ y debido a que j y k son consecutivos, el valor de la función objetivo de ambos calendarios solo se distingue de los valores inducidos por j y k. En S', este valor es $(c+p_k)w_k + (c+p_k+p_j)w_j = cw_k + p_k w_k + cw_j + p_k w_j + p_j w_j$.

Como por hipótesis $p_k < p_j w_k$, se sigue que el valor de la función objetivo de S' es menor que el de S lo cual es una contradicción a la suposición de que S es óptimo. ∎

Ejemplo 1.2. *Antes de terminar esta introducción, consideremos el problema de minimizar la suma de los tiempos de terminación* $(1||\sum \mathbf{C_j})$*, para el cual se deben ordenar los trabajos en orden creciente del tiempo de procesamiento (SPT por sus siglas es inglés).*

Suponiendo que exista un calendario óptimo en el cual al menos dos trabajos no están ordenados por tiempos de procesamiento creciente, podemos suponer como en el teorema anterior que los trabajos son contiguos, es decir, existen pedidos j *y* $j+1$ *para los cuales* $p_j > p_{j+1}$*. Si denotamos por* S_j *el momento en que empieza el pedido* j*, entonces se tiene que* $C_j = S_j + p_j$ *y* $C_{j+1} = S_j + p_j + p_{j+1}$*, pero si intercambiamos los dos trabajos, se tiene* $C_j = S_j + p_{j+1}$ *y* $C_{j+1} = S_j + p_j + p_{j+1}$ *lo cual disminuye el la suma en la cantidad* $p_{j+1} - p_j$*.* □

Los problemas considerados hasta ahora se han podido resolver fácilmente. Más adelante veremos que hay muchos problemas que todavía se pueden resolver, pero de una manera "difícil" (de cuyo significado hablaremos más adelante). El propósito del presente trabajo es establecer una clara línea de división entre los problemas de calendarización "fáciles" ó "difíciles" de resolver.

1.3. Problemas de Optimización y Decisión

De los ejemplos dados anteriormente, se concluye que todo problema de calendarización es básicamente un problema de optimización y en caso extremo hasta un problema de optimización combinatoria cuya complejidad de cálculo resultará evidente a medida que avancemos en nuestro análisis.

El problema de optimización está íntimamente ligado al problema de decisión, del cual haremos uso con frecuencia al analizar la complejidad de un problema de calendarización. Por comparación con uno de optimización, en el cual se busca un valor numérico de la solución; un problema de decisión es uno en el cual se busca una respuesta ó solución del tipo si/no.

La relación entre ambos problemas radica en que un problema de optimización, por ejemplo encontrar un calendario que minimiza $\sum w_j C_j$, se puede resolver mediante la aplicación sucesiva del problema de decisión: dado n ¿existe un calendario para el cual $\sum w_j C_j < n$?

Definición 1.2. *Un* ***problema de decisión*** Π *consta de un conjunto* D_Π *(llamado conjunto de instancias) y un subconjunto* $C \subseteq D_\Pi$ *(llamado el conjunto de instancias o casos SI).*

Los conjuntos D_Π *y* D_Π *son por lo general infinitos y no se dan de manera explícita sino por extensión normalmente.*

Ejemplo 1.3. *El problema* $(1||\sum w_j C_j)$*, el cual recordamos se trata de n trabajos con tiempos de procesamiento* p_j *y peso* w_j*. Si además se establece una cota máxima M a alcanzar, la cuestión de saber si para todas las permutaciones de los trabajos se cumple* $\sum w_j C_j \leq M$*, es claramente un problema* **NP***, pues la única solución es probar todas las permutaciones posibles de los trabajos.* □

Se debe notar que un problema de optimización es por lo menos tan difícil de resolver, como el correspondiente problema de decisión. Pero hacemos referencia a ellos porque son más fáciles de transferir a los conceptos informáticos de computabilidad. A estos conceptos pertenece entre otras cosas el de **lenguajes formales**, de los cuales se hablará en los siguientes capítulos.

Finalmente, para concluir este capítulo, podemos decir que todos los problemas de una sola máquina y n trabajos se podrían resolver de manera heurística probando el valor de la función objetivo para cada una de los $n!$ posibilidades, esto es por supuesto realista en el caso de un valor n pequeño. Como se puede ver en la figura siguiente, el problema resulta intractable para grandes valores de n.

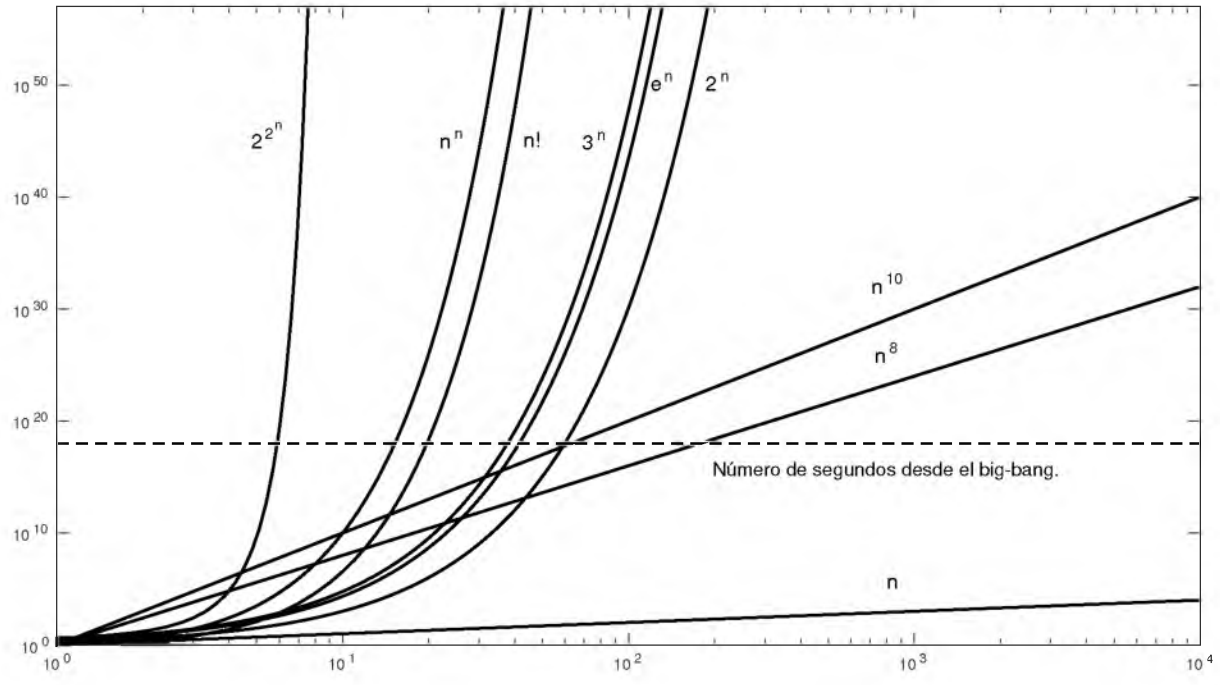


Figura 1.8: Diferentes funciones de crecimiento.

Capítulo 2

Computabilidad

Cada signo por sí mismo parece
muerto; ¿Qué le da vida?

Ludwig Wittgenstein.

2.1. Alfabetos y Cadenas

El mecanismo abstracto que abordaremos en la sección 2.4 es pieza clave para definir la naturaleza de los problemas de decisión. El funcionamiento de tal dispositivo básicamente consiste en procesar cadenas de símbolos de forma sistemática, donde cada una de las cadenas representa una instancia de un problema dado. En este capitulo describiremos los formalismos que conforman a tal dispositivo y que además describen su relación con los problemas de decisión.

Definición 2.1. *Un **alfabeto** $\Sigma = \{a_1, a_2, \ldots, a_n\}$ es un conjunto finito y no vacío, donde cada a_i es llamado símbolo del alfabeto y una **cadena** sobre Σ es una secuencia finita de más de un símbolo o ninguno, esto es, $\omega = b_1 b_2 \cdots b_m$ y además $b_i \in \Sigma$, $i = 1, \ldots, m$.*

Definición 2.2. *Si ω es una cadena sobre un alfabeto finito Σ, entonces se define la **longitud** de ω como el número de posiciones que ocupan los símbolos que la conforman y se denota por $l(\omega)$.*

Las cadenas $\omega_1 = abbbaa$, $\omega_2 = aabaa$ y $\omega_3 = aaa$ formadas del alfabeto finito $\Sigma = \{a, b\}$ tienen por longitudes los valores $l(abbbaa) = 6$, $l(aabaa) = 5$ y $l(aaa) = 3$ respectivamente. En adelante representaremos con las letras griegas $\omega_1, \omega_2, \omega_3, \ldots$ a las cadenas formadas sobre algún alfabeto finito Σ y con la letra griega ε denotaremos a la cadena vacía, la cual no contiene sucesiones de símbolos.

Dadas las cadenas $\omega_1 = a_1 a_2 \ldots a_n$ y $\omega_2 = b_1 b_2 \ldots b_m$ cadenas sobre un alfabeto finito Σ con longitudes n y m respectivamente, se define la operación **concatenación** por $\omega_1 \cdot \omega_2 = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$. Un ejemplo de esta operación, se obtiene de las cadenas $\omega_1 = ababab$ y $\omega_2 = baaabb$ sobre el alfabeto $\Sigma = \{a, b\}$, de donde la cadena $\omega_1 \cdot \omega_2 = ababab \cdot baaabb = abababbaaabb$.

En adición, la definición se puede aplicar a más de dos cadenas, concretamente si consideramos que $\omega_3 = \omega_1 \cdot \omega_2$, obtenemos que la cadena $\omega_3 \cdot \omega_2 = ababababbaaabb \cdot baaabb = abababbaaabbbaaabb$. Por simplicidad, a veces denotaremos $\omega_1\omega_2$ en lugar de $\omega_1 \cdot \omega_2$.

La **potencia de una cadena** es la operación que consiste en concatenar un número finito de veces una cadena arbitraria consigo misma, esta operación se denota por ω^n tal que $\omega^0 = \varepsilon$ y $\omega^n = \omega^1 \cdot \omega^{n-1}$ para $n > 0$. En este caso si $\omega = aabb$ es una cadena sobre el alfabeto $\Sigma = \{a,b\}$, aplicando la definición anterior para un $n = 3$, obtenemos lo siguiente:

$$\begin{aligned}
\omega^0 &= \varepsilon \\
\omega^1 &= \omega^1 \cdot \omega^0 = (aabb) \cdot \varepsilon = aabb \\
\omega^2 &= \omega^1 \cdot \omega^1 = (aabb) \cdot (aabb) = aabbaabb \\
\omega^3 &= \omega^1 \cdot \omega^2 = (aabb) \cdot (aabbaabb) = aabbaabbaabb
\end{aligned}$$

La operación de **cadena inversa** consiste en invertir el orden de los símbolos de una cadena arbitraria, es decir, si $\omega = a_1a_2\cdots a_n$ es una cadena de longitud n, entonces $\omega^R = a_na_{n-1}\cdots a_1$ es la cadena inversa de ω. En este caso, las cadenas $\omega_1 = bababba$ y $\omega_2 = bababbaaabbaba$ sobre el alfabeto $\Sigma = \{a,b\}$, tienen por cadenas inversas a $\omega_1^R = abbabab$ y $\omega_2^R = ababbaaabbabab$ respectivamente.

Con las nociones anteriores se puede definir al conjunto Σ^k que contiene las cadenas de longitud k construidas a partir de un alfabeto finito Σ, donde $k \in \mathbb{N}$. Es importante indicar que para todo alfabeto finito Σ, $\Sigma^0 = \{\varepsilon\}$.

La lista que sigue a continuación exhibe a cada conjunto Σ^k formado del alfabeto $\Sigma = \{a,b\}$,

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \\
\Sigma^1 &= \{a,b\} \\
\Sigma^2 &= \{aa,ab,ba,bb\} \\
\Sigma^3 &= \{aaa,aab,aba,abb,...\} \\
&\vdots
\end{aligned}$$

Al unir todos los conjuntos de la lista anterior, se obtiene el conjunto de todas las cadenas sobre el alfabeto finito Σ, también llamado **el universo de cadenas de** Σ. Esta operación de unir a los conjuntos anteriores se denomina **cerradura de Klenee** y se expresa matemáticamente por $\Sigma^* = \bigcup_{k\geq 0}^{\infty} \Sigma^k$. Por otro lado, la **cerradura más de Kleene** se define como la unión de todos los conjuntos Σ^k excepto el conjunto $\{\varepsilon\}$ y se expresa por $\Sigma^+ = \bigcup_{k\geq 1}^{\infty} \Sigma^k$, donde $k \in \mathbb{N}$.

El universo de cadenas sobre un alfabeto finito Σ tiene la característica de que sus elementos pueden ser enlistados, esto nos permite conocer el tamaño del conjunto Σ^*. Lo anterior será de suma importancia para el desarrollo de los conceptos y resultados que se desarrollarán a lo largo de los siguientes dos capítulos.

Teorema 2.1. *Si Σ es un alfabeto finito, entonces el conjunto Σ^* es numerable.*

Demostración. Se sabe del análisis matemático que la unión numerable de conjuntos finitos es numerable [46, pág. 20]; en este caso la expresión $\Sigma^* = \bigcup_{k\geq 0}^{\infty} \Sigma^k$ es numerable, pues todo conjunto Σ^k es finito. ∎

Una vez establecido que el universo de cadenas sobre un alfabeto finito pertenece a la clase de conjuntos que pueden ser numerados, ahora nos preguntamos ¿Dado el universo de cadenas sobre un alfabeto finito, es el conjunto formado por todos los subconjuntos de cadenas de ese universo un conjunto numerable? Para responder a esta pregunta debemos conocer un par de conceptos al respecto.

2.2. Lenguajes Formales

Un lenguaje formal es un conjunto especial formado de cadenas, sus elementos cumplen alguna regla o propiedad particular, formalmente un lenguaje de define como sigue:

Definición 2.3. *Un lenguaje formal L sobre un alfabeto finito Σ, es un subconjunto finito o infinito de Σ^*, denotado por $L \subseteq \Sigma^*$.*

Por ejemplo, del alfabeto finito $\Sigma = \{a,b\}$ se obtienen los conjuntos $\emptyset$, $\{\varepsilon\}$, $\{a\}$, $\{b\}$ y el propio Σ que son considerados lenguajes triviales de Σ^*. Otros lenguajes que se pueden construir son $\{aaa, aab, abb, baa, bba, bbb\}$, $\{a^n \mid n > 0\}$, $\{\omega\omega^R \mid \omega \in \Sigma^*\}$, $\{a^n b \mid n \geq 0\}$, $\{b^m a \mid m \geq 0\}$ y $\{a^n b^n \mid a > 0\}$ por mencionar algunos ejemplos, en particular el conjunto Σ^* es llamado **el lenguaje universal**.

Las operaciones de concatenación y potencia de cadenas pueden ser extendidas a concepto de lenguajes, en este caso, si L_1 y L_2 son lenguajes en Σ^*, entonces se define la **concatenación** de ambos lenguajes como el conjunto $L_1 \cdot L_2 = \{\omega_1 \cdot \omega_2 \mid \omega_1 \in L_1 \text{ y } \omega_2 \in L_2\}$.

No es muy difícil ver que el lenguaje $\{a^n b^{m+1} a \mid m,n \geq 0\}$ es la concatenación de los lenguajes $\{a^n b \mid n \geq 0\}$ y $\{b^m a \mid m \geq 0\}$. Otro ejemplo es el lenguaje $\{a \cdot \{a,b\}^{m+n} \cdot b \mid m,n \geq 0\}$ resultado de la concatenación del lenguaje $\{a \cdot \{a,b\}^n \mid n > 0\}$ que describe al conjunto de cadenas que inician con el símbolo a seguida de una cadena de longitud n sobre $\{a,b\}$ y el lenguaje $\{\{a,b\}^m \cdot b \mid m > 0\}$ formado por todas las cadenas de longitud m seguidas de un símbolo b.

La **potencia de lenguajes** es la operación que consiste en concatenar n veces consigo mismo un determinado lenguaje L, recursivamente se puede definir como sigue:

$$L^n = \begin{cases} \{\varepsilon\} & \text{si } n = 0, \\ L^1 \cdot L^{n-1} & \text{si } n > 0. \end{cases}$$

En este caso, si consideramos al lenguaje $L = \{0,01\}$ sobre el alfabeto finito $\Sigma = \{0,1\}$, aplicando la definición anterior obtenemos el lenguaje L^4 mediante el siguiente procedimiento:

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= L^1 \cdot L^0 = \{0,01\} \cdot \{\varepsilon\} = \{0,01\} \\ L^2 &= L^1 \cdot L^1 = \{00,001,010,0101\} \\ L^3 &= L^1 \cdot L^2 = \{000,0010,0100,01010,\ldots\} \\ L^4 &= L^1 \cdot L^3 = \{0000,00100,01000,010100,\ldots\} \end{aligned}$$

Otras operaciones simples, pero no menos importantes de los lenguajes son la unión, intersección, diferencia y concatenación. Dados dos lenguajes L_1 y L_2 sobre un alfabeto finito Σ, se define la operación **unión** de L_1 y L_2 como el lenguaje $L_1 \cup L_2 = \{\omega \in \Sigma^* \mid \omega \in L_1 \text{ o } \omega \in L_2\}$, la **intersección** por $L_1 \cap L_2 = \{\omega \in \Sigma^* \mid \omega \in L_1 \text{ y } \omega \in L_2\}$, la **diferencia** por $L_1 - L_2 = \{\omega \in \Sigma^* \mid \omega \in L_1 \text{ y } \omega \notin L_2\}$ y el **complemento** de L_1 como $\overline{L_1} = \{\omega \in \Sigma^* \mid \omega \notin L\}$.

Las operaciones expresadas anteriormente establecen la propiedad **estrella de Kleene** que consiste en unir las todas las potencias[1] de un determinado lenguaje formal, es decir, si L es un lenguaje sobre un alfabeto finito Σ y $L^0, L^1, L^2 \ldots$ son sus respectivas potencias, entonces $L^0 \cup L^1 \cup L^2 \cup \cdots$ es denominado **el lenguaje cerradura de Kleene** y matemáticamente se expresa por $L^* = \bigcup_{k\geq 0}^{n} L^k$. De forma semejante se define **el lenguaje cerradura más de Klenee**, esto es, $L^+ = \bigcup_{k\geq 1}^{n} L^k$.

Consideremos nuevamente al alfabeto $\Sigma = \{a,b\}$ y al lenguaje $L = \{aa,bb\}$, entonces

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= \{aa,bb\} \\ L^2 &= \{aaaa,aabb,bbaa,bbbb\} \\ L^3 &= \{aaaaaa,aaaabb,aabbaa,aabbbb,\ldots\} \\ &\vdots \end{aligned}$$

De modo que

$$\begin{aligned} L^* &= L^0 \cup L^1 \cup L^2 \cup \cdots \\ &= \{\varepsilon,aa,bb,aaaa,aabb,bbaa,\ldots\}. \end{aligned}$$

[1] En particular, se observa que si $L = \Sigma$, el lenguaje L^* coincide con Σ^*.

Con lo expuesto hasta este punto, tenemos un panorama muy general de los lenguajes que se pueden construir sobre un alfabeto finito. Como es de esperar el universo de todos los lenguajes sobre un alfabeto finito Σ, lo conforman todos los subconjuntos posibles de Σ^* y el interés que surge a raíz de este conjunto es saber si es posible categorizarlo de forma disjunta con otros conjuntos conocidos por medio de su cardinalidad tal como se mostró en el teorema 2.1, pues como menciona Abbout: "la cardinalidad proporciona una relación de equivalencia" [47].

Esto último se hace con la intención de estudiar la naturaleza del conjunto con técnicas conocidas o aplicadas a conjuntos de la misma clase. Infortunadamente, el conjunto de todos los lenguajes sobre un alfabeto finito Σ tiene una naturaleza distinta de los conjuntos numerables, tal como se puede apreciar en el siguiente teorema[2].

Teorema 2.2. *El conjunto de todos los lenguajes sobre Σ^* es no numerable.*

Demostración. Sea $S = \{L_0, L_1, L_2, \ldots\}$ el conjunto de todos los lenguajes sobre Σ y supongamos que S es numerable. Del teorema 2.1 sabemos que Σ^* es numerable y en consecuencia sus elementos se pueden enlistar, es decir, $\Sigma^* = \{\omega_0, \omega_1, \omega_2, \ldots\}$.

Construyendo el lenguaje $L_d = \{\omega_i \in \Sigma^* \mid \omega_i \notin L_i\}$, donde $L_i \in S$ y suponiendo que $L_d \in S$, entonces debe existir un $m \in \mathbb{Z}^+$ tal que $L_d = L_m$.

De manera que podemos contemplar los siguientes casos:

1. Si $\omega_m \in L_d$, entonces, de la definición de L_d, se sigue que $\omega_m \notin L_m$, lo cual es una contradicción, debido a que $\omega_m \in L_m$ y $\omega_m \notin L_m$.

2. Si suponemos que $\omega_m \notin L_d$, entonces de la negación de L_d, se sigue que $\omega_m \in L_m$, lo cual nuevamente es una contradicción, debido que $\omega_m \notin L_m$ y $\omega_m \in L_m$.

Por lo anterior concluimos que la hipótesis sobre S es falsa y en consecuencia el conjunto S es no numerable. ∎

La demostración[3] del teorema 2.2 establece que un modelo universal para describir a cada elemento en universo de lenguajes sobre un alfabeto finito Σ es imposible. Las investigaciones de la teoría de la computación se centraron en estudiar modelos matemáticos capaces de especificar la mayor cantidad de lenguajes posibles, esto al menos desde la perspectiva de lenguajes formales, ya que existen diferentes abstracciones con el mismo propósito [37].

Dado que un desarrollo exhaustivo sobre la clasificación de los lenguajes no está dentro de nuestros objetivos. Nos limitaremos unicamente a presentar una breve exposición de los modelos de gramáticas formales con los cuales podemos generar los tipos de lenguajes existentes. Esta exposición tiene única y exclusivamente la intención de poner al lector en contexto con los ejemplos que se usarán en la sección 2.4.

[2] La prueba del teorema 2.2 fue tomada de [37]

[3] El teorema 2.2 utiliza el argumento de diagonalización de George Cantor, para una referencia véase [26].

2.3. Gramáticas Generadoras

Las gramáticas formales o gramáticas generadoras son modelos matemáticos con los cuales podremos generar de forma correcta las cadenas de un determinado lenguaje formal. Estos modelos se deben al lingüista y matemático estadounidense Noam Chomsky quien en los años cuarenta estudió como se adquiría el lenguaje hablado basándose en las reglas de sintaxis de los lenguajes escritos [10].

A continuación presentaremos una exposición amena de los cuadro modelos de Chomsky que describen las diferentes clases de lenguajes existentes, las definiciones son estándar y ampliamente conocidas, las notaciones presentadas aquí fueron tomadas de [36].

2.3.1. Gramáticas no restringidas

Para iniciar con este apartado vamos a definir un par de conceptos fundamentales.

Definición 2.4. *Sea V y Σ alfabetos finitos disjuntos. Una* ***producción*** *sobre V, Σ es una pareja ordenada $(\mu, \upsilon) \in (V \cup \Sigma)^* \cdot V \cdot (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ denotada por la expresión $\mu \to \upsilon$, donde $\mu \in (V \cup \Sigma)^* \cdot V \cdot (V \cup \Sigma)^*$ y $\upsilon \in (V \cup \Sigma)^*$.*

Con la letra P representaremos por al conjunto de producciones $(V \cup \Sigma)^* \cdot V \cdot (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ para facilitar la notación. Los elementos de V se les denomina **variables no terminales** y los elementos de Σ en este caso los llamaremos **símbolos terminales**.

Definición 2.5. *Sea P el conjunto de producciones sobre los alfabetos finitos V, Σ y consideremos a las cadenas $\mu, \upsilon \in (V \cup \Sigma)^*$. Se dice que υ es la derivación en un solo paso de μ si y solo sí existe una producción $A \to \lambda \in P$ tal que $\mu = \alpha_1 A \alpha_2$ y $\upsilon = \alpha_1 \lambda \alpha_2$, y lo abreviamos por $\mu \to \upsilon$, donde $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.*

Por ejemplo, si $CA \to AB$ es una producción sobre los alfabetos $V = \{A, B, C, L, S\}$ y $\Sigma = \{a, b, c\}$, claramente $\upsilon = abbABbCL$ se deriva en un solo paso de $\mu = abbCAbCL$, eligiendo las cadenas $\alpha_1 = abb$ y $\alpha_2 = bCL$, donde $\mu, \upsilon, \alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.

Con los conceptos previos se define el modelo de gramática no restringida como sigue.

Definición 2.6. *Una* ***gramática no restringida*** *es una cuádrupla $G = (V, \Sigma, R, S)$, donde V es un conjunto finito de símbolos no terminales o variables, Σ es un alfabeto finito de símbolos terminales, R es un subconjunto finito de producciones no restringidas sobre V, Σ y $S \in V$ es el símbolo de inicio no terminal.*

La expresión $\mu \to_G^* \upsilon$ denotará a la secuencia de derivaciones $\alpha_i \to_G \alpha_{i+1}$ en un solo paso que transforman a la cadena μ en υ con las reglas de la gramática G. En este caso n representa la longitud de la derivación, tal que $\mu = \alpha_1$, $\upsilon = \alpha_n$ y $\alpha_1, \ldots, \alpha_n \in (V \cup \Sigma)^*$ con $1 < i < n$.

En particular, una cadena $\mu \in (V \cup \Sigma)^*$ se llamará **forma sentencial** de G, si μ se puede derivar desde la variable inicial de G, ésto es, $S \to_G^* \mu$ y una cadena $\omega \in \Sigma^*$ se denomina sentencia sobre G, si existe una derivación $S \to_G^* w$.

El lenguaje generado por una gramática no restringida es denominado **lenguaje del tipo 0** y lo conforman todas las sentencias de la forma $S \rightarrow_G^* \omega$, formalmente se define como sigue:

Definición 2.7. *El lenguaje generado por una gramática no restringida G es el lenguaje $L(G)$ y es definido como $L(G) = \{\omega \in \Sigma^* \mid S \rightarrow_G^* \omega\}$.*

Ejemplo 2.1. *Consideremos al lenguaje $L = \{a^n b^n c^m \mid m \geq 0, n > 0\}$ y a la gramática no restringida $G = \{\{A,B,C,L,S\},\{a,b,c\},R,S\}$, donde R está conformado por el siguiente conjunto de reglas:*

$$\begin{array}{lcl} S & \rightarrow & Sc \\ S & \rightarrow & AB \\ AB & \rightarrow & aABb \\ AB & \rightarrow & ab \end{array}$$

No es difícil establecer que las cadenas del lenguaje L pueden tener la forma $a^n b^n c^m$ y la forma $a^n b^n$ para $m \geq 0$ y $n > 0$ en $\mathbb{Z}^+$. Las cadenas de la forma $a^n b^n c^m$ se pueden obtener de la gramática G aplicando la regla $S \rightarrow Sc$ m veces, una vez $S \rightarrow AB$, $n-1$ veces la regla $AB \rightarrow aABb$ y finalmente una vez la regla $AB \rightarrow ab$ generando la derivación:

$$\begin{array}{lll} S & \rightarrow_G^m & Sc^m \\ & \rightarrow_G & ABc^m \\ & \rightarrow_G^{n-1} & a^{n-1}ABb^{n-1}c^m \\ & \rightarrow_G & a^n b^n c^m \end{array}$$

Análogamente las cadenas de la forma $a^n b^n$ se pueden obtener con un procedimiento parecido, en este caso, aplicando la regla $S \rightarrow AB$ una vez, $n-1$ veces la regla $S \rightarrow aABb$ y una vez la regla $AB \rightarrow ab$ se obtiene la derivación:

$$\begin{array}{lll} S & \rightarrow_G & AB \\ S & \rightarrow_G^{n-1} & a^{n-1}ABb^{n-1} \\ & \rightarrow_G & a^n b^n \end{array}$$

De esta manera todas las cadenas que siguen los patrones $a^n b^n c^m$ y $a^n b^n$ en el lenguaje L se pueden construir con las reglas de la gramática G propuesta. □

Ejemplo 2.2. *Sea $L = \{a^n b^n c^n \mid n > 0\}$ un lenguaje y sea $G = \{\{S,L\},\{a,b,c\},R,S\}$ una gramática no restringida, donde R está conformado por el siguiente conjunto de reglas:*

$$\begin{array}{lcl} S & \rightarrow & aScb \\ S & \rightarrow & L \\ cb & \rightarrow & bc \\ Lb & \rightarrow & b \end{array}$$

Para generar las cadenas del lenguaje L, se observa que la cadena abc se genera con la derivación $S \to_G aLcb \to_G aLbc \to_G abc$. *Para generar las cadenas de la forma* $a^n b^n c^n$ *cuando* $n > 0$ *se aplica en primera instancia la regla* $S \to aScb$ *n veces y una vez la regla* $S \to L$, *obteniendo así la derivación,*

$$\begin{aligned} S &\to_G^* a^n S(cb)^n \\ &\to_G a^n L(cb)^n \end{aligned}$$

Con la secuencia anterior obtenemos los n símbolos a, b y c respectivamente desordenados. A partir de la última regla se aplica la regla $cb \to bc$ *una vez al símbolo b más a la izquierda, el siguiente símbolo b más a la izquierda requiere de dos veces la aplicación de la regla* $cb \to bc$ *y así sucesivamente hasta llegar al n-enésimo símbolo b que requiere de n veces la aplicación de la regla antes mencionada.*

Finalmente se aplica una vez la regla $Lb \to b$ *obteniendo la secuencia ordenada, lo anterior se ilustra en la siguiente derivación,*

$$\begin{aligned} S &\to_G a^n Lcbcbcb\cdots cb \\ S &\to_G a^n Lbccbcb\cdots cb \\ S &\to_G a^n Lbcbccb\cdots cb \\ S &\to_G a^n Lbbcccb\cdots cb \\ &\vdots \\ S &\to_G a^n Lbb^{n-1}c^n \\ S &\to_G a^n b^n c^n \end{aligned}$$

Siguiendo este patrón podemos generar las cadenas en $\{a^n b^n c^n \mid n > 0\}$.

En particular, la cadena $aaabbbcc \in \{a^n b^n c^n \mid n > 0\}$ *se puede generar aplicando la siguiente secuencia de reglas:*

$$\begin{aligned} S &\to_G a\mathbf{S}cb \\ S &\to_G aa\mathbf{S}cbcb \\ S &\to_G aaa\mathbf{S}cbcbcb \\ S &\to_G aaaL\mathbf{cb}cbcb \\ S &\to_G aaaLbc\mathbf{cb}cb \\ S &\to_G aaaLb\mathbf{cb}ccb \\ S &\to_G aaaLbbcc\mathbf{cb} \\ S &\to_G aaaLbbc\mathbf{cb}c \\ S &\to_G aaaLbb\mathbf{cb}cc \\ S &\to_G aaa\mathbf{Lb}bbccc \\ S &\to_G aaabbbccc \end{aligned}$$

□

El aspecto principal del modelo no restringido es la libertad que existe para definir las reglas de producción, observemos que la generación de la cadena *aabbcc* se realiza por medio de frases, por este motivo el modelo no restringido es denominado también **modelo estructurado por frases**.

2.3.2. Gramáticas sensibles al contexto

El conjunto de producciones de una gramática sensible al contexto es exactamente el tipo de producciones definidas en 2.4, con la excepción de que dado una pareja (a,b) en el conjunto de producciones R se cumple que b tiene al menos la misma longitud que a, es decir, $l(a) \leq l(b)$, formalmente la gramática se describe a continuación:

Definición 2.8. *Una **gramática sensible al contexto** es una cuádrupla $G = (V,\Sigma,R,S)$, donde V es un conjunto finito de símbolos no terminales o variables, Σ es un alfabeto finito de símbolos terminales, R es un subconjunto finito de $(V \cup \Sigma)^* \cdot V \cdot (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ tal que cada producción en R de la forma $\alpha A \beta \to \alpha \lambda \beta$, implica que $l(\alpha A) \leq l(\alpha \lambda \beta)$, donde $A \in V$, $\alpha, \lambda, \beta \in (V \cup \Sigma)^*$, $\lambda \neq \varepsilon$ y $S \in V$ es el símbolo de inicio no terminal.*

El conjunto de producciones de una gramática sensible al contexto puede contener la producción $S \to \varepsilon$ siempre que el lenguaje a generar contenga a la cadena vacía. Dado que la restricción de una gramática sensible al contexto consiste en elegir reglas de producción acotadas como se definió anteriormente, las nociones de producción en un solo paso, derivaciones y el lenguaje generado son las mismas que en el modelo no restringido. Con respecto a esto último el lenguaje generado por una gramática sensible al contexto es conocido como **lenguaje del tipo 1**.

Ejemplo 2.3. *Sea $G = (\{B,S\},\{a,b,c\},R,S)$ una gramática sensible al contexto donde R es el conjunto de producciones siguientes*

$$\begin{array}{lcl} S & \to & aSBc \\ S & \to & abc \\ cB & \to & Bc \\ bB & \to & bb \\ B & \to & b \end{array}$$

No es difícil establecer el esquema general con el cual podemos formar las cadenas del lenguaje $L = \{a^n b^n c^n \mid n > 0\}$ con las reglas de la gramática G, por ejemplo si elegimos la cadena $a^3b^3c^3$ puede ser generada por la siguiente derivación,

$$\begin{array}{lcl} S & \to_G & aSBc \\ & \to_G & aaSBcBc \\ & \to_G & aaabcBcBc \\ & \to_G & aaabBccBc \\ & \to_G & aaabBcBcc \\ & \to_G & aaabBBccc \\ & \to_G & aaabbBccc \\ & \to_G & aaabbbccc \end{array}$$

□

En el ejemplo 2.3 se definió una gramática que genera al lenguaje $\{a^n b^n c^n \mid n > 0\}$, tres de las cuatro reglas que conforman esa gramática mantienen la condición necesaria para ser una gramática sensible al contexto con la excepción de la regla $Lb \to b$, donde $|Lb| > |b|$ y en consecuencia esa gramática no puede ser sensible al contexto.

2.3.3. Gramáticas libres del contexto

La gramática libre del contexto es otro modelo de gramática con restricciones, en este caso las producciones de este modelo tienen una estructura particular que se presenta en la siguiente definición.

Definición 2.9. *Una **producción libre del contexto** sobre los alfabetos disjuntos V, Σ es una pareja ordenada $(u,v) \in V \times (V \cup \Sigma)^*$ expresada por $u \to v$ donde $u \in V$ y $v \in (V \cup \Sigma)^*$.*

De esta definición se puede observar que el conjunto de producciones libres del contexto es en realidad un subconjunto del conjunto de producciones P sobre los alfabetos finitos V y Σ, es decir, $V \times (V \cup \Sigma)^* \subset (V \cup \Sigma)^* \cdot V \cdot (V \cup \Sigma)^* \times (V \cup \Sigma)^*$. Por lo tanto, el lenguaje generado por la gramática libre del contexto se define exactamente como se ha definido en los modelos anteriores y es conocido como **lenguaje del tipo 2**.

Ejemplo 2.4. *Sea $G = (\{S\}, \{a,b\}, \{S \to aSb, S \to ab\}, S)$ una gramática libre del contexto, entonces con el siguiente esquema la gramática G genera las cadenas del lenguaje $L = \{a^n b^n \mid n > 0\}$ sobre el alfabeto $\{a,b\}$.*

Para ello nos percatamos de que la cadena ab se genera con la regla $S \to ab$ y que todas las cadenas de la forma $a^n b^n \in L$ para $n > 0$, se obtienen aplicando $n-1$ veces la regla $S \to aSb$ y una vez la regla $S \to ab$ obteniendo la derivación,

$$\begin{aligned} S &\to_G^{n-1} a^{n-1} S b^{n-1} \\ &\to_G a^n b^n \end{aligned}$$

Siguiendo este esquema se pueden generar todas las cadenas del lenguaje $\{a^n b^n \mid n > 0\}$. □

Ejemplo 2.5. *La gramática $G = (\{a,b\}, \{S\}, \{S \to \varepsilon \mid a \mid b, S \to aSa \mid bSb\}, S)$ una gramática libre del contexto puede generar las cadenas del lenguaje $L = \{\omega \in \Sigma^* \mid \omega = \omega^R\}$ y la gramática $G = (\{a,b\}, \{S\}, \{S \to A, S \to a \mid AAAB, B \to b\}, S)$ libre del contexto puede generar las cadenas del lenguaje $L = \{a^m b^n \mid m = 2n+1$ con $m,n \in \mathbb{Z}^+\}$.* □

2.3.4. Gramáticas regulares

El modelo de gramática regular genera los **lenguajes del tipo 3** y se considera también un tipo de gramática restringida. Este modelo es el más simple de todos los modelos que especifican lenguajes y sus reglas de producción tienen la siguiente estructura.

Definición 2.10. *Sea V y Σ un par de alfabetos finitos disjuntos. Una* ***producción regular*** *en V, Σ es una pareja ordenada $(u,v) \in V \times \Sigma^* \cdot (V \cup \varepsilon)$ o $(u,v) \in V \times \Sigma^* \cdot (V \cup \varepsilon)$ denotada por la expresión $u \to v$ donde $u \in V$ y $v \in \Sigma^* \cdot (V \cup \varepsilon)$ o $v \in V \times \Sigma^* \cdot (V \cup \varepsilon)$.*

Al igual que en los modelos anteriores no es difícil ver que los conjuntos $V \times \Sigma^* \cdot (V \cup \varepsilon)$ y $V \times (V \cup \varepsilon) \cdot \Sigma^*$ son subconjuntos del conjunto de producción libre del contexto $V \times (V \cup \Sigma)^*$, de modo que el lenguaje generado por una gramática regular es un lenguaje libre del contexto.

Definición 2.11. *Una* ***gramática regular*** *es una cuádrupla $G = (V, \Sigma, R, S)$, donde V es un conjunto finito de símbolos no terminales o variables, Σ es un alfabeto finito de símbolos terminales, R es un subconjunto finito de $V \times \Sigma^* \cdot (V \cup \varepsilon)$ denominado reglas de producción y $S \in V$ es llamado símbolo de inicio.*

Ejemplo 2.6. *Sea $G = (\{S\}, \{a\}, \{S \to aS, S \to a\}, S)$ una gramática regular, entonces la gramática G genera al lenguaje $\{a^n \mid n > 0\}$. Particularmente las cadenas $\omega = aa$, $\omega = aaa$ y $\omega = aaaaaaa$ pueden ser generadas por las siguientes derivaciones,*

$$
\begin{array}{lll}
S & \to_G & aS \\
 & \to_G & aaS \\
 & \to_G & aaa
\end{array}
\qquad
\begin{array}{lll}
S & \to_G & aS \\
 & \to_G & aaS \\
 & \to_G & aaaS \\
 & \to_G & aaaaS \\
 & \to_G & aaaaa
\end{array}
\qquad
\begin{array}{lll}
S & \to_G & aS \\
 & \to_G & aaS \\
 & \to_G & aaaS \\
 & \to_G & aaaaS \\
 & \to_G & aaaaaS \\
 & \to_G & aaaaaaS \\
 & \to_G & aaaaaaa
\end{array}
$$

□

Ejemplo 2.7. *La gramática regular $G = (\{A,S\}, \{a,b\}, \{S \to BS \mid \varepsilon, B \to AA, A \to a \mid b\}, S)$ genera al lenguaje $\{\omega \in \Sigma^* \mid |\omega| = 2k, k \in \mathbb{Z}^+\}$. En particular las cadenas $\omega = bbba$, $\omega = aabb$ y $\omega = aaab$ son generadas por las siguientes derivaciones,*

$$
\begin{array}{lll}
S & \to_G & BS \\
 & \to_G & AABS \\
 & \to_G & AAAAS \\
 & \to_G & bAAAS \\
 & \to_G & bbAAS \\
 & \to_G & bbbAS \\
 & \to_G & bbbaS \\
 & \to_G & bbba\varepsilon \\
 & \to_G & bbba
\end{array}
\qquad
\begin{array}{lll}
S & \to_G & BS \\
 & \to_G & AABS \\
 & \to_G & AAAAS \\
 & \to_G & aAAAS \\
 & \to_G & aaAAS \\
 & \to_G & aabAS \\
 & \to_G & aabbS \\
 & \to_G & aabb\varepsilon \\
 & \to_G & aabb
\end{array}
\qquad
\begin{array}{lll}
S & \to_G & BS \\
 & \to_G & AABS \\
 & \to_G & AAAAS \\
 & \to_G & aAAAS \\
 & \to_G & aaAAS \\
 & \to_G & aaaAS \\
 & \to_G & aaabS \\
 & \to_G & aaab\varepsilon \\
 & \to_G & aaab
\end{array}
$$

□

2.3.5. La jerarquía de Chomsky

Los cuadros modelos matemáticos presentados previamente describen las cuatro clases importantes de lenguajes que pueden ser estudiadas dentro del universo de lenguajes sobre un alfabeto finito Σ, estos modelos constituyen la conocida jerarquía Chomsky (véase, [36]), la cual establece que cada clase de lenguaje está contenida propiamente una dentro de otra, empezando por la clase de lenguajes del tipo 3 y finalizando con la clase de lenguaje del tipo 0, esto es,

$$L_{t_3} \subset L_{t_2} \subset L_{t_1} \subset L_{t_0}.$$

Donde L_{t_0}, L_{t_1}, L_{t_2} y L_{t_3} representan a la clase de lenguajes del tipo 0, del tipo 1, del tipo 2 y del tipo 3 respectivamente. Es importante decir que la contención anterior es estrictamente incluyente, pues no todos los elementos en L_{t_0} están incluidos en L_{t_1} y así sucesivamente.

La clasificación de Chomsky fue la conclusión de importantes lineas de investigación sobre varios modelos de computación propuestos a lo largo de los años, entre ellos tenemos la máquina de Turing, autómatas finitos, autómatas de pila y autómatas linealmente acotados. A cada clase de lenguaje en la jerarquía de Chomsky se le asocia un mecanismo abstracto de reconocimiento [40]. La tabla 2.1 muestra un resumen de las clases de lenguajes existentes con sus respectivos modelos de gramática y su mecanismo de reconocimiento asociado.

Lenguajes	**Gramática**	**Mecanismo**
Tipo 0	No restringida	Máquina de Turing
Tipo 1	Sensible al contexto	Autómata linealmente acotado
Tipo 2	Libre al contexto	Autómata de pila
Tipo 3	Regular	Autómata finito

Tabla 2.1: Jerarquía de Chomsky.

Para establecer los fundamentos teóricos de la jerarquía de Chomsky se requiere de una gran cantidad de detalles formales al respecto, la demostración de esta jerarquía se puede considerar como el desarrollo de la teoría de computación en sí misma. Si el lector desea comprender los detalles de la clasificación de la tabla 2.1, puede consultar los primeros seis capítulos del libro de Jhon E. Hopcroft y Jeffrey D. Ullman (1979) para una descripción amena de los mecanismos de autómata finito y de pila junto con sus equivalencias con los modelos de gramáticas asociados.

El autómata linealmente acotado está asociado a la noción no determinista que veremos en el capítulo siguiente. Una vez comprendida la noción del modelo de Turing y el concepto no determinista es fácil comprender el funcionamiento de este modelo. Al respecto se puede consultar el trabajo de Martin Davis (véase, [36]), una fuente avanzada es el trabajo de G. Rozenberg y A. Salomaa (1997) (véase, [39]) que además es una buena fuente de consulta sobre los modelos de gramáticas.

Como hemos expresado al final de la sección 2.2, nuestro interés es dar un panorama general de la teoría de los lenguajes formales para poner en contexto a los lectores menos expertos y familiarizarlos con los ejemplos de la siguiente sección. También porque la noción de "problema de decisión" es definida mediante el mecanismo más general de cómputo que existe hasta nuestros días que es la máquina de Turing y además porque la complejidad computacional que es el tema central del trabajo se define sobre este mecanismo.

Es importante mencionar que el mecanismo de Turing que fue propuesto con la intención de formalizar la idea de "Procedimiento efectivo" introducido por David Hilbert (véase, [2]) y que dio paso a la clasificación presentada en la tabla 2.1. A continuación daremos una exposición del mecanismo de Turing y de sus nociones que le acompañan.

2.4. Modelo computacional de Turing

En esta sección se presenta una breve exposición del modelo de cómputo abstracto introducido por el matemático británico Alan M. Turing[4] en 1936 para resolver "el problema de decisión" o "Entscheidungsproblem" enunciado por David Hilbert en 1928 [6]. El modelo que presentaremos a continuación es equivalente al modelo original de Alan Turing con algunas sutiles diferencias, basado principalmente en las definiciones presentadas en [41] y [37].

2.4.1. Descripción y funcionamiento del modelo de Turing

Una máquina de Turing es un modelo matemático que describe el funcionamiento mecánico de un dispositivo abstracto que procesa símbolos sobre una cinta infinita dividida por celdas, donde cada una de las celdas contiene algún espacio en blanco o símbolo perteneciente a un alfabeto finito. La cinta se extiende indefinidamente por la derecha y está limitada por el extremo izquierdo.

El procesamiento de los símbolos sobre la cinta se realiza mediante una unidad de control conformada por un número finito de elementos denominados "estados" y una cabeza lectora de símbolos con la capacidad adicional de escribir. La figura 2.1 muestra una posible descripción gráfica del dispositivo abstracto de Turing.

[4]La vida y obra de Alan Madison Turing se puede encontrar en [63] o en [64].

Formalmente, una máquina de Turing se representa con el siguiente modelo matemático:

Definición 2.12. *Una* Máquina de Turing *M estándar es una 6-tupla* $(Q, \Sigma, \Gamma, \delta, q_0, F)$*, donde:*

- Q *es un conjunto finito de* estados,
- Σ *es un* alfabeto *finito,*
- Γ *es un* alfabeto *de la cinta,*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, S\}$ *es la* función de transición,
- $q_0 \in Q$ *es el* estado inicial, *y*
- $F \subseteq Q$ *es el conjunto de* estados finales o de detención.

El alfabeto finito Σ es un subconjunto $\Gamma - \{\triangleright\}$, el símbolo $\triangleright \in \Gamma$ representa un espacio vacío sobre la cinta de la máquina de Turing. La función de transición δ es un conjunto de reglas que establecen las acciones realizadas por la unidad de control y la cabeza de lectura/escritura, denotadas por $\delta(p, \sigma) = (q, \gamma, D)$ con $p, q \in Q$; $\sigma, \gamma \in \Gamma$ y $D \in \{R, L, S\}$. Las letras R, L y S indican un movimiento a la derecha, izquierda y movimiento neutro.

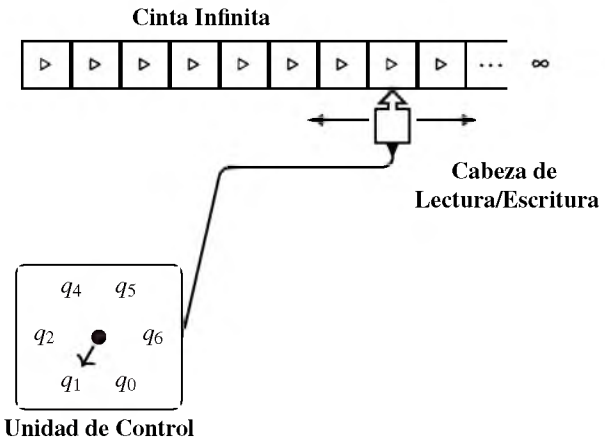
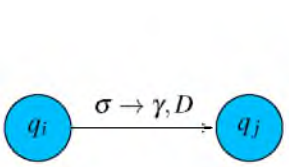


Figura 2.1: Máquina de Turing

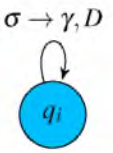
Una transición de una máquina Turing realiza las siguientes acciones:

1. La unidad control se encuentra en el "estado actual" p.
2. En el estado actual p se realiza la lectura del símbolo σ y se reescribe por γ.
3. Se realiza un movimiento en dirección a D y la unidad de control toma un "nuevo estado" q.

Las transiciones de una máquina de Turing se pueden representar pictóricamente por una gráfica denominada "diagrama de transición", por ejemplo la transición $\delta(q_i,\sigma) = (q_j,\gamma,D)$ se representa por la gráfica (a) y la transición de la forma $\delta(q_i,\sigma) = (q_i,\gamma,D)$, cuando $q_i = q_j$ se representa con la gráfica (b), donde $i, j \in \mathbb{N}$.



(a) Diagrama 1



(b) Diagrama 2

Ejemplo 2.8. *Sea $M_{D1} = (\{q_0,q_1,q_2\},\{a,b\},\{a,b,X,Y,\triangleright\},\delta,q_0,F=\emptyset)$ un modelo de máquina de Turing, donde la función de transición $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R,L,S\}$ se define con las siguientes reglas:*

$$\begin{array}{lcllcl} \delta(q_0,\triangleright) & = & (q_1,\triangleright,R) & \delta(q_2,b) & = & (q_1,Y,R) \\ \delta(q_1,a) & = & (q_1,X,R) & \delta(q_2,X) & = & (q_1,X,R) \\ \delta(q_1,b) & = & (q_2,b,L) & & & \end{array}$$

Por comodidad se usa una tabla para colocar las transiciones de una máquina de Turing, donde los elementos del conjunto Q disponen de las filas y los elementos de Γ de las columnas. La tabla de la izquierda corresponde a las transiciones de la máquina M_{D1} y en la figura de la derecha se presenta su correspondiente diagrama de transición.

δ	$\triangleright$	a	b	X	Y
q_0	$(q_1,\triangleright,R)$	–	–	–	–
q_1	–	(q_1,X,R)	(q_2,b,L)	–	–
q_2	–	–	(q_1,Y,R)	(q_1,X,R)	–

Tabla 2.2: Transiciones de M_{D1}

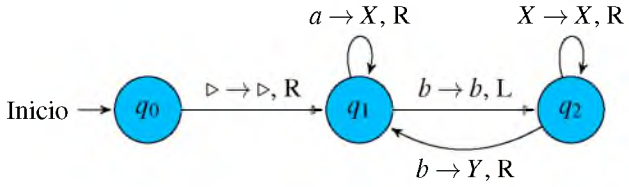


Figura 2.2: Diagrama de transición de M_{D1}

□

Los espacios vacíos marcados por el guión en la tabla del ejemplo anterior, indican que la regla de transición no esta definida para la correspondiente fila y columna. Por convención, cuando $\delta(q,\sigma)$ no esté definida para algún $q \in Q$ y $\sigma \in \Gamma$, diremos que la máquina de Turing se detiene.

Definición 2.13. *Una configuración o descripción instantánea sobre la cinta de una máquina de Turing M es una cadena uqv, donde $q \in Q$, y $u,v \in \Gamma^*$.*

Las cadenas sobre la cinta de una máquina de Turing estarán limitadas por espacios en blanco, el primer espacio se encuentra en la primera celda de la cinta y al final de la palabra hay una cantidad infinita de espacios en blanco, tal como se aprecia en la figura 2.3. En cada instante del procesamiento de los símbolos sobre la cinta se genera un registro denominado **descripción instantánea** o **configuración**. En este caso la configuración de la figura 2.3 se escribe $\triangleright\, \omega_1 q_j \omega_2 \,\triangleright$, donde $\omega_1 = \sigma_1\sigma_2$ y $\omega_2 = \sigma_3\sigma_4\sigma_5\sigma_6$.

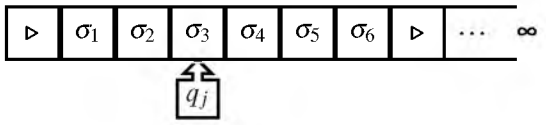


Figura 2.3: Configuración sobre la cinta.

La configuración de inicio de una máquina de Turing M es una cadena de la forma q_0u, donde $u \in \Gamma^*$ y $q_0 \in Q$. La generación de configuraciones al aplicar una regla de transición se denomina paso de configuración y se denota por $\vdash_M$, obteniéndose tres posibles casos para $\sigma(q_i, \sigma)$ definida:

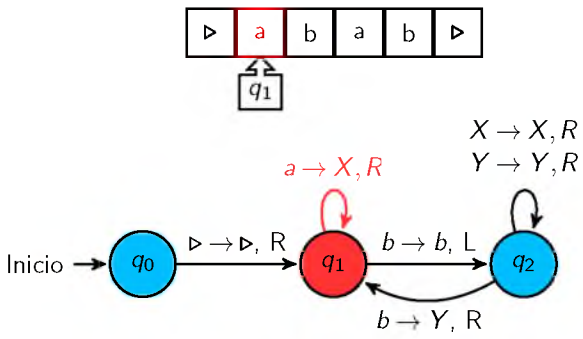
- Si $\delta(q_i, \sigma) = (q_j, \gamma, R)$, entonces $uq_i\sigma v \vdash_M u\gamma q_j v$;
- Si $\delta(q_i, \sigma) = (q_j, \gamma, L)$, entonces $u\sigma q_i v \vdash_M uq_j\gamma v$; y
- Si $\delta(q_i, \sigma) = (q_j, \gamma, S)$, entonces $uq_i\sigma v \vdash_M uq_j\gamma v$.

En cada uno de los casos las cadenas $u, v \in \Gamma^*$; $q_i, q_j \in Q$ y $\sigma, \gamma \in \Gamma$ con $i, j \in \mathbb{N}$.

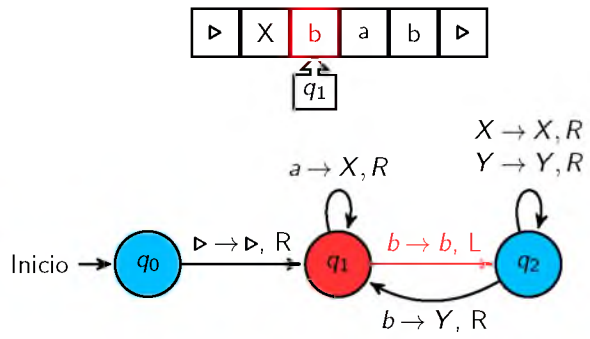
Definición 2.14. *Sea M una máquina de Turing estándar y ω una cadena. La computación de M sobre ω es una secuencia de configuraciones $C_0, C_1, C_2, \cdots, C_n$, para algún $n \geq 0$ que conduce a la detención de M tal que $C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_n$. Si n existe, el tamaño de la computación se representa por $\tau_M(\omega) = n$ y si la computación es infinita se representa por $\tau_M(\omega) = \infty$.*

Las siguiente secuencia de imágenes de (a) hasta (e), ilustran paso a paso el funcionamiento de la máquina de Turing M_{D1} del ejemplo 2.8, mostrando los respectivos movimientos de la cabeza lectora sobre la cinta infinita al computar la cadena de entrada $\triangleright abab \triangleright$, partiendo de la configuración $\triangleright q_1 abab \triangleright$.

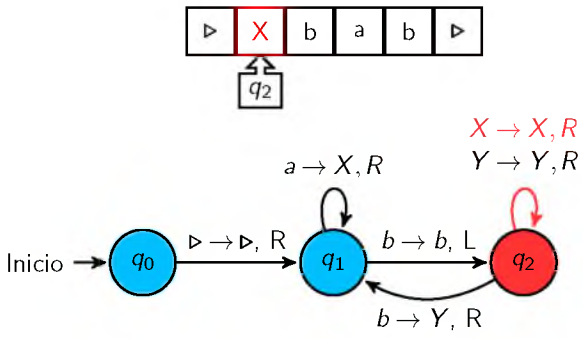
La configuración $\triangleright q_1 abab \triangleright$ representa la situación mostrada en la figura (a), al aplicar la transición $\delta(q_1, a) = (q_1, X, R)$ remarcada con rojo, se obtiene la configuración mostrada en la cinta de la figura (b) provocando que la cabeza se mueva a la derecha. El paso de la configuración de la figura (b) a la configuración de la figura (c) provoca un movimiento de la cabeza hacia la izquierda.



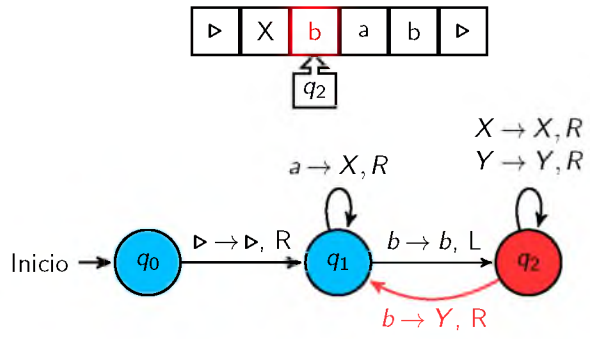
(a)



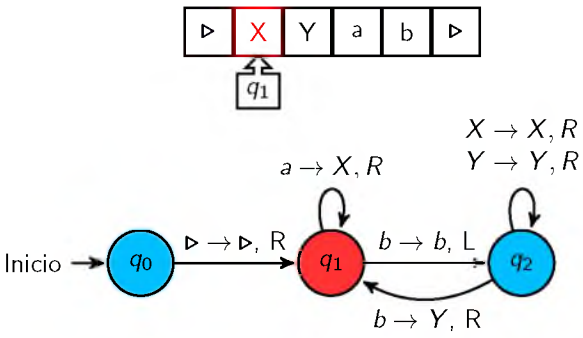
(b)



(c)



(d)



(e)

Dado que $\delta(q_1, X)$ no esta definida, las secuencia de descripciones instantáneas obtenidas en las figuras anteriores terminan en la configuración de detención $\triangleright\, q_1 XYab \triangleright$, obteniéndose el siguiente registro:

$$\begin{aligned} \triangleright\, q_1 abab \triangleright &\vdash_M \triangleright\, X q_1 bab \triangleright \\ &\vdash_M \triangleright\, q_2 X bab \triangleright \\ &\vdash_M \triangleright\, X q_2 bab \triangleright \\ &\vdash_M \triangleright\, q_1 XYab \triangleright \end{aligned}$$

Para abreviar las secuencias de configuraciones obtenidas por la aplicación finita de transiciones se usa la expresión $\vdash_M^*$. De esta manera podemos abreviar a la secuencia de configuraciones anteriores por $\triangleright\, q_1 abab \triangleright \vdash_M^* \triangleright\, q_1 XYab \triangleright$.

En el próximo apartado se definirá la detención de la máquina de Turing por estado final, de momento vamos a hablar sutilmente de una noción muy importante que provoca que una maquina de Turing no se detenga a causa de un "bucle infinito".

En la máquina de "bucle infinito" la cabeza de lectura/escritura se mantiene en un movimiento perpetuo de izquierda a derecha y viceversa impidiendo que la máquina se detenga; el siguiente diagrama de transición corresponde a una máquina de Turing que se cicla al computar la subcadena *ab*.

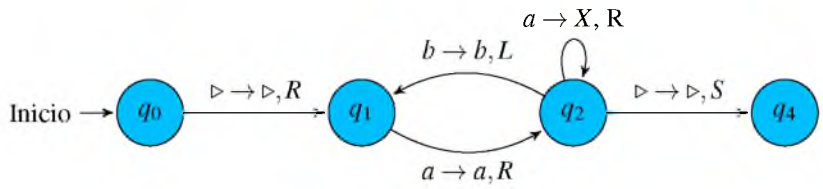


Figura 2.4: Máquina de Turing con bucle infinito.

En la secuencia de configuraciones, $q_0 \triangleright abaa \triangleright \vdash_M \triangleright q_1 abaa \triangleright \vdash_M \triangleright aq_2 baa \triangleright \vdash_M \triangleright q_1 abaa \triangleright \vdash_M^* \cdots$ se aprecia el prolongamiento infinito del paso de la configuración $\triangleright q_1 abaa \triangleright$ a la configuración $\triangleright aq_2 baa \triangleright$ y luego en sentido contrario, esta situación la abreviaremos con la expresión $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \infty$.

2.4.2. El lenguaje reconocido y lenguaje decidido por una máquina de Turing

La clase de lenguajes del tipo 0 mencionados en la sección 2.3.5 corresponden a todos aquellos lenguajes que pueden ser reconocidos por una máquina de Turing M y son denominados lenguajes **recursivos enumerables**.

Definición 2.15. *Sea* $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ *una máquina de Turing. El lenguaje reconocido por* M *es* $L(M) = \{\omega \in \Sigma^* \mid q_0\omega \vdash_M^* \omega_1 q_f \omega_2\}$, *donde* $q_f \in F$ *y* $\omega_1, \omega_2 \in \Gamma^*$.

De la definición observamos que la aceptación de una cadena ω sucede siempre que la computación de la máquina de Turing empiece desde la configuración inicial y terminé en una configuración de detención en el estado final. De esta manera una cadena ω será rechazada por una máquina de Turing cuando "la computación de la máquina se detenga en un estado no final o cuando la máquina no se detenga a causa de un bucle infinito" [51].

Sin embargo, si L es un lenguaje sobre Σ y $\omega \notin L$, la definición 2.15 no implica que la máquina de Turing M rechace a la cadena ω, en otras palabras, la máquina de Turing M puede aceptar cadenas que no están en el lenguaje [61]. Por otro lado, si un lenguaje es recursivo enumerable y cumple la definición 2.16 deteniéndose para toda entrada, entonces recibe el nombre de **lenguaje recursivo** o **lenguaje decidible**.

Definición 2.16. *Sea $L \subseteq \Sigma^*$ un lenguaje y M una maquina de Turing que se detiene para toda cadena de entrada. Decimos que M decide L si M acepta a ω para toda $\omega \in L$ y M rechaza a ω para toda $\omega \notin L$.*

A continuación presentaremos algunos ejemplos clásicos de lenguajes y sus respectivos modelos de máquinas de Turing frecuentes en la literatura con la intención de establecer familiaridad con el funcionamiento del mecanismo abstracto.

Ejemplo 2.9. *Sean $L_{a^n b^m} = \{a^n b^m \mid m,n > 0\}$ y $L_{a^n b^n} = \{a^n b^n \mid n > 0\}$ lenguajes y consideremos a la máquina de Turing $M_{a^n b^m} = (\{q_0,q_1,q_2,q_3,q_4,q_5\},\{a,b\},\{a,b,X,Y,\triangleright\},\delta,\{q_0\},\{q_4\})$, donde la función de transición δ se define en la tabla 2.3.*

δ	$\triangleright$	a	b	X	Y
q_0	$(q_1,\triangleright,R)$	–	–	–	–
q_1	–	(q_2,X,R)	(q_5,Y,R)	–	–
q_2	–	(q_2,X,R)	(q_3,Y,R)	–	–
q_3	$(q_4,\triangleright,S)$	(q_5,X,R)	(q_3,Y,R)	–	–
q_4	–	–	–	–	–
q_5	–	(q_5,X,R)	(q_5,Y,R)	–	–

Tabla 2.3: Transiciones de $M_{a^n b^m}$

En el siguiente diagrama de transición que corresponde a la máquina de Turing $M_{a^n b^m}$ se puede corroborar que la computación de cualquier cadena de la forma $a^n b^m$ siempre termina en el estado final de aceptación q_4 y cualquier cadena que no tenga la estructura anterior es rechazada. En particular, la computación de las cadenas aaabbb y aaaabbbb terminan en el estado q_4, tal como se aprecia en el registro de la tabla 2.4.

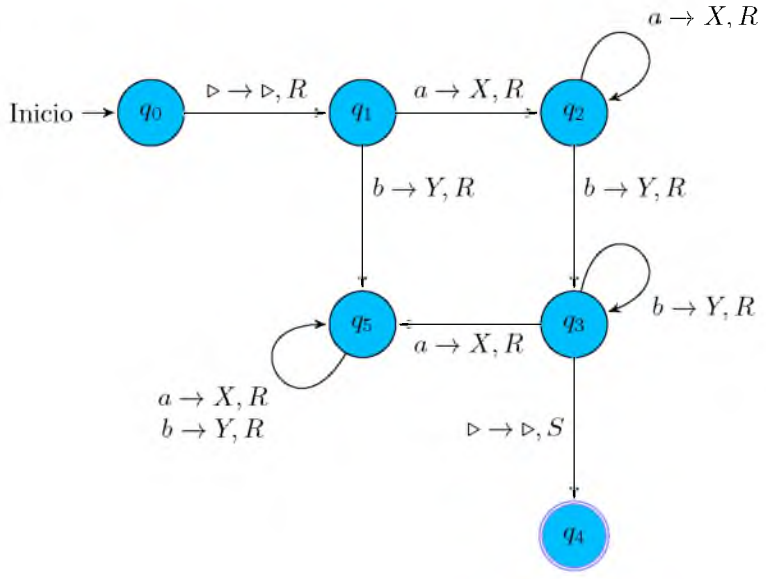


Figura 2.5: Diagrama de transición de $M_{a^n b^m}$.

Por otro lado, la máquina de Turing $M_{a^n b^m}$ reconoce al lenguaje $L_{a^n b^n}$ pues cualquier cadena de la forma $a^n b^n$ siempre terminará en el estado final q_4, sin embargo la máquina $M_{a^n b^m}$ no decide al lenguaje $L_{a^n b^n}$, en particular podemos ver que la cadena aaabbbb no es un elemento del lenguaje $L_{a^n b^n}$.

Computación de *aaabbb*		Computación de *aaaabbbb*	
$q_0 \triangleright a\,a\,a\,b\,b\,b\,\triangleright$	$\vdash_M \triangleright\, q_1\, a\,a\,a\,b\,b\,b\,\triangleright$	$q_0 \triangleright\, a\,a\,a\,a\,b\,b\,b\,b\,\triangleright$	$\vdash_M \triangleright\, q_1\, a\,a\,a\,a\,b\,b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, X\, q_2\, a\,a\,b\,b\,b\,\triangleright$		$\vdash_M \triangleright\, X q_2\, a\,a\,a\,b\,b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, XX\, q_2\, a\,b\,b\,b\,\triangleright$		$\vdash_M \triangleright\, XX q_2\, a\,a\,b\,b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, XXX q_2\, b\,b\,b\,\triangleright$		$\vdash_M \triangleright\, XXX q_2\, a\,b\,b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, XXXY q_3\, b\,b\,\triangleright$		$\vdash_M \triangleright\, XXXX q_2\, b\,b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, XXXYY q_3\, b\,\triangleright$		$\vdash_M \triangleright\, XXXXY q_3\, b\,b\,b\,\triangleright$
	$\vdash_M \triangleright\, XXXYYY q_3\,\triangleright$		$\vdash_M \triangleright\, XXXXYY q_3\, b\,b\,\triangleright$
	$\vdash_M \triangleright\, XXXYYY q_4\,\triangleright$		$\vdash_M \triangleright\, XXXXYYY q_3\, b\,\triangleright$
			$\vdash_M \triangleright\, XXXXYYYY q_3\,\triangleright$
			$\vdash_M \triangleright\, XXXXYYYY q_4\,\triangleright$

Tabla 2.4: Computación de *aaabbb* y *aaaabbbb*.

□

Ejemplo 2.10. *En contraste con lo anterior, el lenguaje $L_{a^n b^n} = \{a^n b^n \mid n > 0\}$ es decidido por el modelo de máquina de Turing $M_{a^n b^n} = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \{a, b, X, Y, \triangleright\}, \delta, \{q_5\})$ con la función de transición δ definida en la tabla 2.5 y su respectivo diagrama de transición representado en la figura 2.6.*

δ	$\triangleright$	a	b	X	Y
q_0	$(q_1, \triangleright, R)$	–	–	–	–
q_1	–	(q_2, X, R)	–	–	(q_4, Y, R)
q_2	–	(q_2, a, R)	(q_3, b, L)	–	(q_2, Y, R)
q_3	–	(q_3, a, L)	–	(q_1, X, R)	(q_3, Y, L)
q_4	$(q_5, \triangleright, S)$	–	–	–	(q_4, Y, R)
q_5	–	–	–	–	–

Tabla 2.5: Transiciones de $M_{a^n b^n}$.

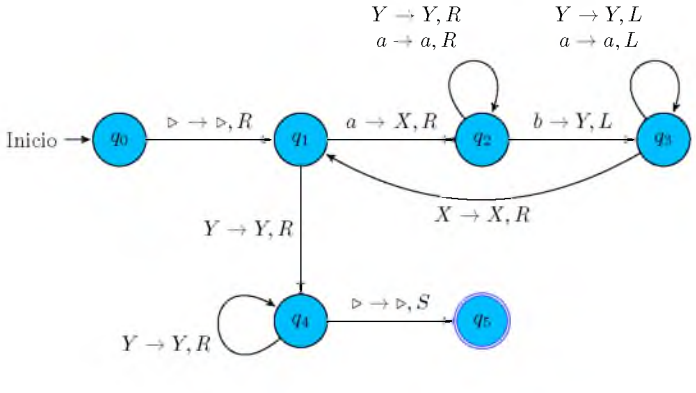


Figura 2.6: Diagrama de transición de $M_{a^n b^n}$.

En la tabla 2.6 se presenta el proceso seguido por el mecanismo $M_{a^n b^n}$ al aceptar la cadena aaabbb y rechazar la cadena aaabb.

Computación de *aaabbb*			Computación de *aaabb*		
$q_0 \triangleright\ a\ a\ a\ b\ b\ b \triangleright$	$\vdash_M$	$\triangleright\, q_1\ a\ a\ a\ b\ b\ b \triangleright$	$q_0 \triangleright aaabb \triangleright$	$\vdash_M$	$\triangleright\, q_1\ a\ a\ a\ b\ b \triangleright$
	$\vdash_M$	$\triangleright\, X q_2\ a\ a\ b\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X q_2\ a\ a\ b\ b \triangleright$
	$\vdash_M$	$\triangleright\, X a\ q_2\ a\ b\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X a\ q_2\ a\ b\ b \triangleright$
	$\vdash_M$	$\triangleright\, X a\ a\ q_2\ b\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X a\ a\ q_2\ b\ b \triangleright$
	$\vdash_M$	$\triangleright\, X a\ q_3\ a\ Y\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X a\ q_3\ a\ Y\ b \triangleright$
	$\vdash_M$	$\triangleright\, X q_3\ a\ a\ Y\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X q_3\ a\ a\ Y\ b \triangleright$
	$\vdash_M$	$\triangleright\, q_3 X\ a\ a\ Y\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, q_3 X\ a\ a\ Y\ b \triangleright$
	$\vdash_M$	$\triangleright\, X q_1\ a\ a\ Y\ b\ b \triangleright$		$\vdots$	
	$\vdash_M$	$\triangleright\, X X q_2\ a\ Y\ b\ b \triangleright$		$\vdash_M$	$\triangleright\, X X a\ q_3 Y Y \triangleright$
	$\vdots$			$\vdash_M$	$\triangleright\, X X q_3\ a Y Y \triangleright$
	$\vdash_M$	$\triangleright\, X X X q_1 Y Y Y \triangleright$		$\vdash_M$	$\triangleright\, X q_3 X\ a Y Y \triangleright$
	$\vdash_M$	$\triangleright\, X X X Y q_4 Y Y \triangleright$		$\vdash_M$	$\triangleright\, X X q_1\ a Y Y \triangleright$
	$\vdash_M$	$\triangleright\, X X X Y Y q_4 Y \triangleright$		$\vdash_M$	$\triangleright\, X X X q_2 Y Y \triangleright$
	$\vdash_M$	$\triangleright\, X X X Y Y Y q_4 \triangleright$		$\vdash_M$	$\triangleright\, X X X Y q_2 Y \triangleright$
	$\vdash_M$	$\triangleright\, X X X Y Y Y q_5 \triangleright$		$\vdash_M$	$\triangleright\, X X X Y Y q_2 \triangleright$

Tabla 2.6: Comparación de computaciones.

□

Ejemplo 2.11. *El lenguaje* $L_{a^n b^n c^n} = \{a^n b^n c^n \mid n > 0\}$ *es decidido por la máquina de Turing* $M_{a^n b^n c^n} = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b\}, \{a, b, X, Y, Z, \triangleright\}, \delta, \{q_6\})$ *con la función de transición* δ *de la siguiente tabla.*

δ	$\triangleright$	a	b	c	X	Y	Z
q_0	(q_1, B, R)	–	–	–	–	–	–
q_1	–	(q_2, X, R)	–	–	–	(q_5, Y, R)	–
q_2	–	(q_2, a, R)	(q_3, Y, R)	–	–	(q_2, Y, R)	–
q_3	–	–	(q_3, b, R)	(q_4, Z, L)	–	–	(q_3, Z, R)
q_4	–	(q_4, a, L)	(q_4, b, L)	–	(q_1, X, L)	(q_4, Y, L)	(q_4, Z, L)
q_5	$(q_6, \triangleright, S)$	–	–	–	–	(q_5, Y, R)	(q_5, Z, R)
q_6	–	–	–	–	–	–	–

Tabla 2.7: Diagrama de transición de $M_{a^n b^n c^n}$

El diagrama de transición la máquina de Turing M_3 *es:*

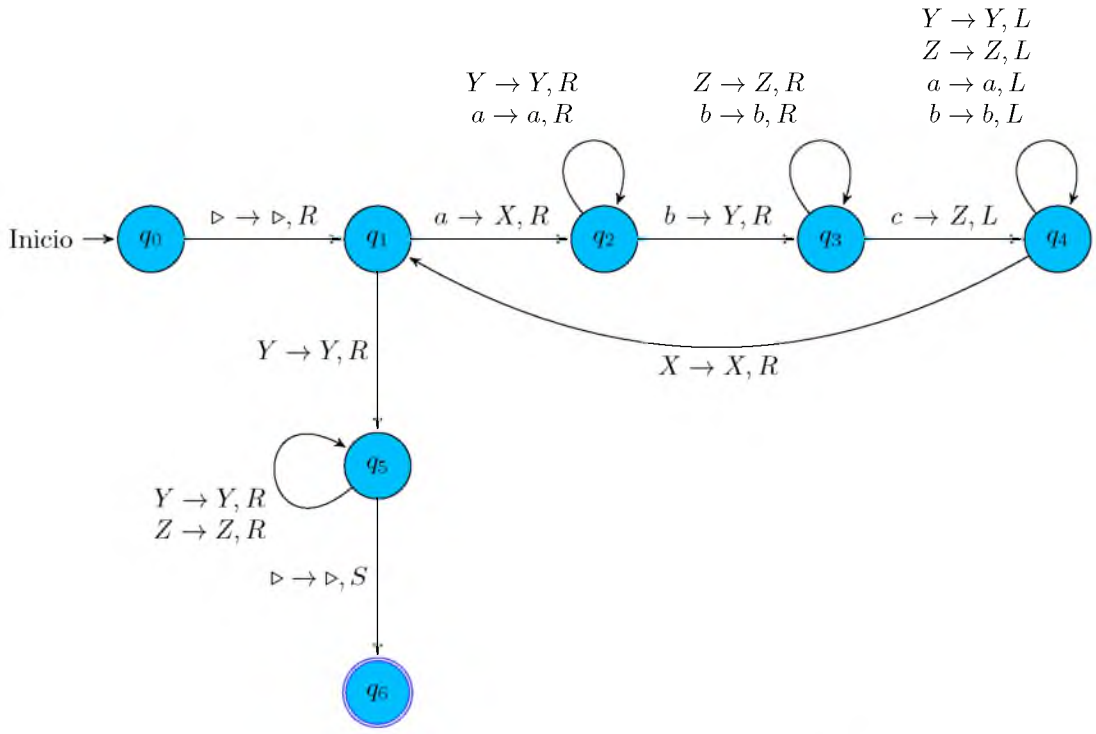


Figura 2.7: Diagrama de transición de $M_{a^n b^n c^n}$.

Las cadenas aceptadas en el conjunto $\{a^n b^n c^n \mid n > 0\}$ *llegan al estado* q_6. □

2.4.3. Composición de máquinas de Turing

En algunas ocasiones el proceso de reconocimiento de cadenas de un determinado lenguaje es laborioso. La naturaleza del mecanismo de Turing nos permite fusionar más de una máquina de Turing para resolver un problema determinado. Esta importante característica la describiremos a continuación y será aplicada en el problema del apartado 2.5.1.

Definición 2.17. *Sean* $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, s_1, F_1)$ *y* $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, s_2, F_2)$ *dos máquinas de Turing sobre el alfabeto de entrada* Σ *y el alfabeto de la cinta* Γ *tal que* $Q_1 \cap Q_2 = \emptyset$. *La máquina de Turing* $M = (Q, \Sigma, \Gamma, \delta, s, F)$ *es la composición de* M_1 *y* M_2 *denotada por* $M = M_1 M_2$ *donde* $Q_1 \cup Q_2 = Q$, $s = s_1$, $F = F_2$ *y*

$$\delta(q,\sigma) = \begin{cases} \delta_1(q,\sigma) & \text{si } q \in Q_1 \text{ y } \delta_1(q,\sigma) \neq (q',\tau,D) \text{ para todo } q' \in F_1, \\ (s_2,\sigma,D) & \text{si } q \in Q_1 \text{ y } \delta_1(q,\sigma) = (q',\tau,D) \text{ para algún } q' \in F_1, \\ \delta_2(q,\sigma) & \text{si } q \in Q_2. \end{cases}$$

Ejemplo 2.12. *Sea máquina de Turing* $M_{C1} = (\{q_0,q_1,q_2,q_3,q_4,q_5\}, \{a\}, \{a, X, \triangleright\}, \delta_1, q_0, \{q_5\})$ *con el siguiente diagrama de transición* δ_1,

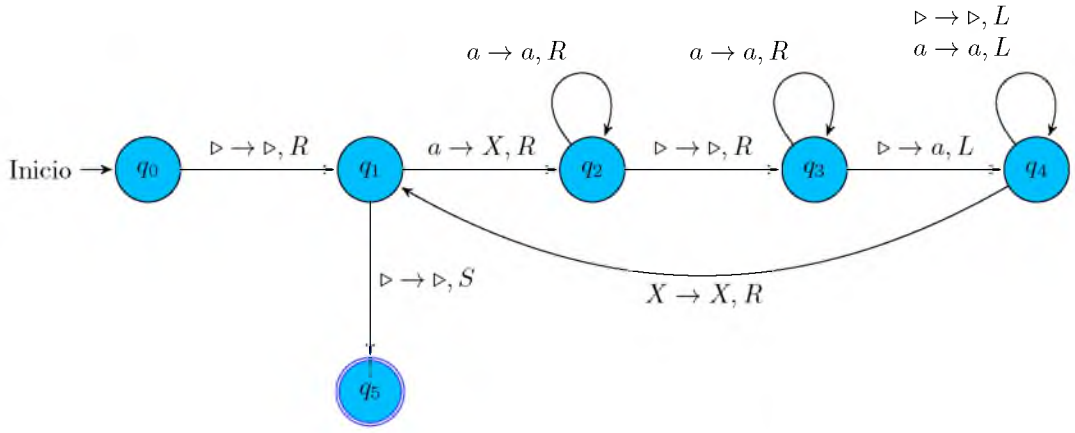


Figura 2.8: Diagrama de transición de M_{C1}.

La máquina de Turing M_{C1} *genera una copia de la cadena de entrada sobre el alfabeto finito* Σ *separando la copia de la cadena por medio de un espacio en blanco* $\triangleright$. *La máquina cambia por X los primeros símbolos hasta el primer espacio en blanco* $\triangleright$ *tal como se ilustra en la computación de la cadena* $\triangleright\, a\, a\, \triangleright$,

$$
\begin{array}{lll}
q_0 \triangleright a\,a \triangleright \triangleright \triangleright \triangleright & \vdash_M & \triangleright q_1\,a\,a \triangleright \triangleright \triangleright \triangleright \\
& \vdash_M & \triangleright X\,q_2\,a \triangleright \triangleright \triangleright \triangleright \\
& \vdash_M & \triangleright X\,a\,q_2 \triangleright \triangleright \triangleright \triangleright \\
& \vdash_M & \triangleright X\,a \triangleright\, q_3 \triangleright \triangleright \triangleright \\
& \vdash_M & \triangleright X\,a\,q_4 \triangleright\, a \triangleright \triangleright \\
& \vdash_M & \triangleright X\,q_4\,a \triangleright\, a \triangleright \triangleright \\
& \vdash_M & \triangleright q_4\,X\,a \triangleright\, a \triangleright \triangleright \\
& \vdash_M & \triangleright X\,q_1\,a \triangleright\, a \triangleright \triangleright
\end{array}
\qquad
\begin{array}{lll}
\triangleright X\,q_1\,a \triangleright\, a \triangleright \triangleright & \vdash_M & \triangleright X\,X\,q_2 \triangleright a \triangleright \triangleright \\
& \vdash_M & \triangleright X\,X \triangleright\, q_3 a \triangleright \triangleright \\
& \vdash_M & \triangleright X\,X \triangleright a\,q_3 \triangleright \triangleright \\
& \vdash_M & \triangleright X\,X \triangleright\, q_4\,a\,a \triangleright \\
& \vdash_M & \triangleright X\,X\,q_4 \triangleright\, a\,a \triangleright \\
& \vdash_M & \triangleright X\,q_4\,X \triangleright\, a\,a \triangleright \\
& \vdash_M & \triangleright X\,X\,q_1 \triangleright\, a\,a \triangleright \\
& \vdash_M & \triangleright X\,X\,q_5 \triangleright\, a\,a \triangleright
\end{array}
$$

Es importante decir que la máquina M_{C1} procesa cadenas $\triangleright\ \triangleright \cdots \triangleright$ que terminan en el estado final y para los cuales la máquina no realiza una copia, para el buen funcionamiento del mecanismo las entradas siempre deben ser cadenas de la forma $\triangleright\, a\, a \cdots a \triangleright$ sobre el alfabeto Σ. Por otro lado, consideremos a la máquina de Turing $M_{C2} = (\{p_0, p_1, p_2\}, \{a\}, \{a, X, \triangleright\}, \delta_2, p_0, \{p_3\})$ con la función de transición presentada en el diagrama siguiente.

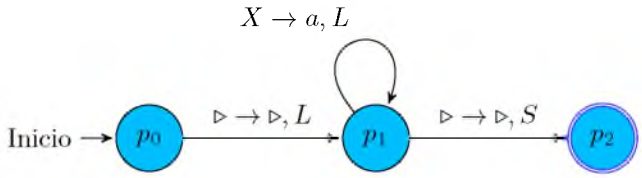


Figura 2.9: Diagrama de transición de M_{C2}.

Esta máquina de Turing M_{C2} transforma los símbolos X en símbolos a que se encuentren más a la izquierda empezando sus operaciones desde la última configuración generada por la máquina M_{C1}. El siguiente registro se obtiene del cómputo de la máquina M_{C2} al procesar la configuración $\triangleright XXq_5 \triangleright\, a\, a \triangleright$ del registro anterior cambiando el estado q_5 por p_0.

$$
\begin{array}{lll}
\triangleright X\,X\,p_0 \triangleright\, a\,a \triangleright & \vdash_M & \triangleright X p_1\,X \triangleright\, a\,a \triangleright \\
& \vdash_M & \triangleright\, p_1\,X\,a \triangleright\, a\,a \triangleright \\
& \vdash_M & p_1 \triangleright\, a\,a \triangleright\, a\,a \triangleright \\
& \vdash_M & p_2 \triangleright\, a\,a \triangleright\, a\,a \triangleright
\end{array}
$$

El modelo $M_C = (\{q_0, q_1, q_2, q_3, q_4, p_0, p_1, p_2\}, \{a\}, \{a, X, \triangleright\}, \delta, q_0, \{p_2\})$ se obtiene al componer los mecanismos M_{C1} y M_{C2} formando una máquina copiadora, donde δ está conformada por el siguiente conjunto de reglas obtenidas de aplicar la definición 2.17,

$$
\begin{array}{llll}
\delta(q_0,\triangleright) = (q_1,\triangleright,R) & \delta(q_3,a) = (q_3,a,R) & \delta(p_0,\triangleright) = (p_1,\triangleright,L) \\
\delta(q_1,a) = (q_2,X,R) & \delta(q_3,\triangleright) = (q_4,a,L) & \delta(p_1,X) = (p_1,a,L) \\
\delta(q_1,\triangleright) = (p_0,\triangleright,S) & \delta(q_4,\triangleright) = (q_4,\triangleright,L) & \delta(p_1,\triangleright) = (p_2,\triangleright,S) \\
\delta(q_2,a) = (q_2,a,R) & \delta(q_4,a) = (q_4,a,L) & \\
\delta(q_2,\triangleright) = (q_3,\triangleright,R) & \delta(q_4,X) = (q_1,X,R) &
\end{array}
$$

El diagrama de transición para las reglas anteriores se muestra a continuación. En el diagrama observamos que el estado q_5 es remplazado por p_0, ya que es en este punto donde comienza la computación de la máquina M_{C2}.

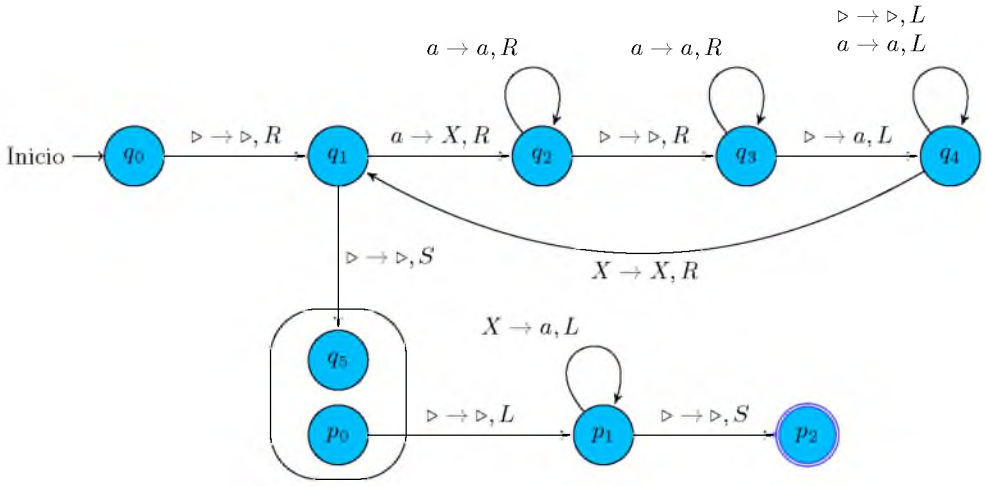


Figura 2.10: Diagrama de transición M_C

□

2.5. Indecibilidad y el poder de cómputo

En el contexto de la máquinas de Turing un lenguaje es indecidible, si el lenguajes es recursivo enumerable, pero no recursivo. En este caso solo podemos demostrar que existe una máquina de Turing que reconoce al lenguaje, pero no una máquina de Turing que lo decida [43]. El problema de detención es el primer problema con esta característica que dio paso al descubrimiento de más problemas de esta índole. Para entender la naturaleza del problema de detención será necesario describir la codificación de máquinas de Turing.

Definición 2.18. *Sea $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ una máquina de Turing con el alfabeto $\Gamma = \{\sigma_1, \sigma_2, \ldots\}$ y el conjunto de estados $Q = \{q_0, q_1, q_2, \ldots\}$ finitos. Consideremos a $N_\Gamma(\sigma_i) = i$ como la función que asigna números a los elementos de Γ y a $N_Q(q_j) = j+1$ como la función que asigna números a los elementos de Q, donde $i, j \in \mathbb{Z}^+$. Las direcciones R, L, S de la máquina serán asignadas por $N_D(R) = 1$, $N_D(L) = 2$ y $N_D(S) = 3$.*

De lo anterior, cada transición T de la forma $\delta(q_k, \sigma_l,) = (q_r, \sigma_s, D)$ de la máquina de Turing M se representará con la función $C(T) = 0^k 10^l 10^r 10^s 10^{N(D)}$. Con estos detalles se define la codificación completa de la máquina de Turing M se representa como la función

$$\psi(M) = 111C(T_1)11C(T_2)11\cdots 11C(T_n)111,$$

donde $T_1, T_2, T_3, \ldots, T_n$ las transiciones ordenadas de M. Finalmente, si $\omega = \sigma_1\sigma_2\ldots\sigma_m$ es una cadena sobre Σ, entonces $c(\omega) = 0^{N(\sigma_1)}10^{N(\sigma_2)}1\cdots 10^{N(\sigma_m)}$ es la codificación binaria de la cadena ω cuando $\Sigma \neq \{0,1\}$.

Ejemplo 2.13. *Consideremos las transiciones de la máquina de Turing $M_{a^nb^n}$ del ejemplo 2.10, donde $\Gamma = \{a,b,X,Y,\triangleright\}$ y $Q = \{q_0,q_1,q_2,q_3,q_4,q_5\}$. Al aplicar la definición anterior obtenemos la siguientes codificaciones.*

$$
\begin{array}{lcccll}
T_1 : \delta(q_0,\triangleright) & = & (q_1,\triangleright,R) & \rightarrow & C(T_1) & = & 0^1 10^1 10^2 10^1 10^1 \\
T_2 : \delta(q_1,a) & = & (q_2,X,R) & \rightarrow & C(T_2) & = & 0^2 10^2 10^3 10^4 10^1 \\
T_3 : \delta(q_1,Y) & = & (q_4,Y,R) & \rightarrow & C(T_3) & = & 0^2 10^5 10^5 10^5 10^1 \\
T_4 : \delta(q_2,a) & = & (q_2,a,R) & \rightarrow & C(T_4) & = & 0^3 10^2 10^3 10^2 10^1 \\
T_5 : \delta(q_2,b) & = & (q_3,b,L) & \rightarrow & C(T_5) & = & 0^3 10^3 10^4 10^3 10^2 \\
T_6 : \delta(q_2,Y) & = & (q_2,Y,R) & \rightarrow & C(T_6) & = & 0^3 10^5 10^3 10^5 10^1 \\
T_7 : \delta(q_3,a) & = & (q_3,a,L) & \rightarrow & C(T_7) & = & 0^4 10^2 10^4 10^2 10^2 \\
T_8 : \delta(q_3,X) & = & (q_1,X,R) & \rightarrow & C(T_8) & = & 0^4 10^4 10^2 10^4 10^1 \\
T_9 : \delta(q_3,Y) & = & (q_3,Y,L) & \rightarrow & C(T_9) & = & 0^4 10^5 10^4 10^5 10^2 \\
T_{10} : \delta(q_4,\triangleright) & = & (q_5,\triangleright,S) & \rightarrow & C(T_{10}) & = & 0^5 10^1 10^6 10^1 10^3 \\
T_{11} : \delta(q_4,Y) & = & (q_4,Y,R) & \rightarrow & C(T_{11}) & = & 0^5 10^5 10^5 10^5 10^1
\end{array}
$$

Finalmente la codificación de la máquina M_2 es :

$$\psi(M_2) = 111C(T_1)11C(T_2)11C(T_3)11C(T_4)11\cdots 11C(T_9)11C(T_{10})11C(T_{11})111$$

En el ejemplo 2.10 obtuvimos la computación de la cadena $\omega = aaabbb$ sobre $\Sigma = \{a,b\}$, aplicando la definición de codificación para ω obtenemos que:

$$c(\omega) = 0^1 10^1 10^1 10^2 10^2 10^2$$

□

2.5.1. El problema de detención de máquinas de Turing

En todos los ejemplos anteriores, observamos que las salidas de las máquinas de Turing pueden ser vistas como respuestas del tipo si/no, por resultado de procesar una instancia que pertenece o no al lenguaje para el cual fueron diseñadas.

Problemas de diversas disciplinas como teoría de las gráficas, teoría de números o la teoría de juegos, entre otras, pueden ser representados como problemas de reconocimiento de lenguajes mediante la codificación adecuada de sus instancias por medio de una cadena sobre un alfabeto finito [16].

Aunque esta idea es fascinante, no todos los problemas representados en términos de lenguajes proporcionan una respuesta de tipo si/no. En este caso, el problema de detención es uno entre muchos problemas con esta característica.

En el problema de detención se plantea: Dado una máquina de Turing M_D arbitraria y una cadena ω arbitraria sobre un alfabeto finito Σ ¿Es posible determinar si la maquina M_D se detiene con la cadena ω? Para responder esta pregunta debemos representar el problema en términos de lenguajes, en este caso, la pregunta anterior corresponde a una instancia del conjunto de máquinas de Turing que tienen la propiedad de detenerse ante una cadena de entrada. Este lenguaje se representa por $L_h = \{\psi(M)c(\omega) \mid M \text{ se detiene } \omega\}$, donde $\psi(M)c(\omega)$ es la concatenación binaria de la codificación de la máquina M y una cadena ω arbitraria.

El siguiente teorema establece que no existe una máquina de Turing M_D capaz de decidir los elementos del lenguaje L_h.

Teorema 2.3. *El lenguaje $L_h = \{\psi(M)c(\omega) \mid M$ se detiene $\omega\}$ es indecidible.*

Demostración. Supongamos que el problema de detención para máquinas de Turing es decidible, es decir, existe una máquina de Turing M_H tal que,

$$M_H(\psi(M)\omega) = \begin{cases} acepta & \text{si } M \text{ se detiene con } \omega, \\ rechaza & \text{si } M \text{ no se detiene con } \omega. \end{cases}$$

La expresión $\psi(M)\omega$ es la concatenación conformada por la codificación binaria de una máquina de Turing arbitraria M seguida de la codificación binaria de una cadena $\omega \in \Sigma^*$.

A partir de la máquina de Turing M_H construimos una máquina de Turing $M_{H'}$ que recibe de entrada a la cadena $\psi(M)$ y se comporta exactamente como M_H con la excepción de que al llegar al estado de aceptación, la máquina de Turing $M_{H'}$ entra en un bucle infinito, es decir, si q_f es el estado de aceptación de la máquina de Turing M_H, entonces al terminar el cómputo en el estado q_f, la máquina $M_{H'}$ entra en un ciclo como el que se observa en la figura 2.11.

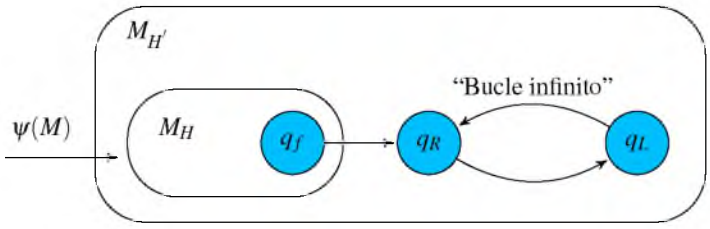


Figura 2.11: Composición de $M_{D'}$.

Al combinar la máquina de Turing $M_{H'}$ con una máquina copiadora M_C que recibe como entrada a una cadena ω y genera una salida $\omega\omega$, ésto es, $M_C(\omega) = \omega \cdot \omega$ obtendremos la máquina de Turing compuesta $E = M_{H'}M_C$ como se ilustra en la figura 2.12.

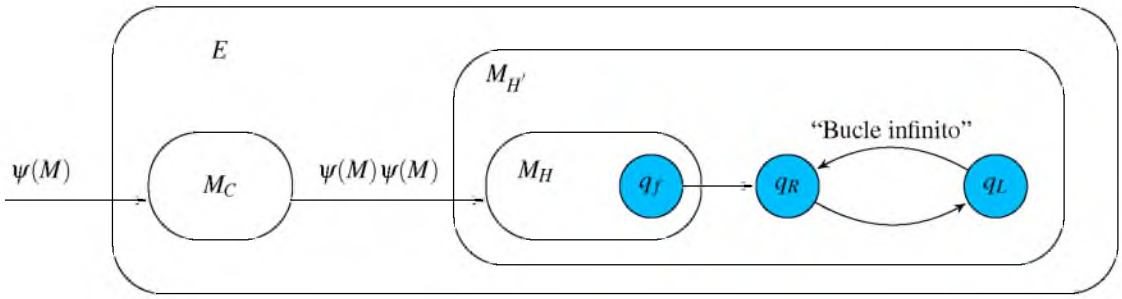


Figura 2.12: Composición de E.

Si $\psi(M)$ es una codificación binaria de una máquina de Turing M arbitraria, entonces

$$\begin{aligned} E(\psi(M)) &= M_{H'}(M_C(\psi(M))) \\ &= M_{H'}(\psi(M)\psi(M)) \\ &= \begin{cases} cicla & \text{si } M \text{ se detiene con } \psi(M), \\ rechaza & \text{si } M \text{ no se detiene con } \psi(M). \end{cases} \end{aligned}$$

En particular, E es una máquina de Turing y puede ser codificada en una cadena binaria $\psi(E)$. Si consideramos la computación de la máquina E con su propia codificación $\psi(E)$, observamos que si E se detiene con $\psi(E)$, entonces E no se detiene con $\psi(E)$, lo cual implica una contradicción. Por otro lado, si E no se detiene con $\psi(E)$, entonces E se detiene con $\psi(E)$, lo cual nuevamente es una contradicción.

Por este razonamiento, concluimos que E no puede existir y como consecuencia $M_{H'}$ no existe, ya que E se construyó a partir de $M_{H'}$. Por el mismo argumento M_H no existe y por lo tanto el problema de detención es indecidible. ∎

La demostración para establecer si L_h es recursivo enumerable corresponde a la construcción de una máquina de Turing que lo reconozca. El diseño presentado en la definición del modelo 2.12, conlleva ciertas complicaciones, sin embargo, el problema puede ser resuelto fácilmente por medio de la construcción de una máquina de k-cintas como la definida en el capítulo 3, con este mecanismo se simula el proceso llevado a cabo por los estados de la máquina arbitraria a partir de la cadena de entrada. Desde luego el proceso es largo y no lo presentaremos aquí.

Una exposición detallada sobre la construcción de la máquina universal usada para reconocer al lenguaje L_h se encuentra en el capitulo 7 del libro de Minsky [12] y la simulación de la máquina que reconoce al lenguaje L_h se encuentra en el capítulo 11 de [41].

2.5.2. El lenguaje no recursivo enumerable

El problema de detención abrió paso al descubrimiento de más problemas de la misma naturaleza, entre ellos encontramos el problema de correspondencia de Post, el problema de la equivalencia de las gramáticas libres de contexto, la ambigüedad de las gramáticas libres del contexto (véase, capítulo 9 de [61]), el décimo problema de Hilbert (véase, [34]), entre otros. Aunque estos problemas son indecidibles, existe un mecanismo para cada uno de ellos capaz de reconocerlos.

La pregunta que surge a partir de lo anterior es ¿Existe un limite en la capacidad de reconocimiento del mecanismo de Turing? La respuesta es que sí, para verlo debemos considerar al universo de todas las máquinas de Turing que tienen como alfabeto de entrada al conjunto binario $\Sigma = \{0,1\}$. Con la técnica de codificación antes presentada es posible asociar una cadena binaria a cada máquina de Turing con estas características.

Si aplicamos el orden lexicográfico al conjunto de todas las cadenas binarias, nos podemos percatar de que algunas cadenas no corresponden a la codificación de una máquina de Turing y también pueden existir muchos posibles códigos para una solo máquina de Turing. Esto nos indica que el conjunto de máquinas de Turing codificadas es más pequeño que el conjunto de cadenas binarias, en otras palabras podemos enlistar las máquinas de Turing mediante una lista infinita, $M_1, M_2, M_3, \ldots$ con más de una cadena binaria ω_j ($j = 1,2,3,\ldots$) asociada [23].

Con lo anterior estamos en condiciones de construir el lenguaje $L_d = \{\omega_i \in \{0,1\}^* \mid \omega_i \notin L(M_i)\}$, el cual lo conforman todas las cadenas binarias w_i tal que la máquina M_i rechaza a w_i cuando ésta es su propia codificación.

Teorema 2.4. *Sea $L_d = \{\omega_i \in \{0,1\}^* \mid \omega_i \notin L(M_i)\}$, entonces no existe una máquina de Turing M que reconozca a L_d.*

Demostración. Supongamos que L_d es recursivo enumerable, entonces L_d debe ser reconocido por una máquina de Turing M, tal que $L_d = L(M)$.

Dado que sabemos que las codificaciones de máquinas de Turing pueden ser enumeradas mediante una lista $M_1, M_2, M_3, \ldots$ y que L_d es un lenguaje sobre el alfabeto finito $\Sigma = \{0,1\}$, entonces la máquina de Turing M debe ser parte de la lista de máquina de Turing codificadas, es decir, existe un $i \in \mathbb{Z}$ tal que $M = M_i$, de modo que $L_d = L(M_i)$.

Si $\omega_i \in L_d$, entonces $\omega_i \notin L(M_i)$, lo cual es una contradicción, ya que ω_i es y no es un elemento en $L(M)$. Por otro lado, si $\omega_i \in L_d$, entonces de la negación de L_d, obtenemos que $\omega_i \in L(M_i)$, lo cual también es una contradicción, pues $L_d = L(M)$.

De lo anterior concluimos que el lenguaje L_d no es recursivo enumerable y en consecuencia no existe una máquina de Turing que lo reconozca. ∎

Con la demostración del teorema anterior se establece el limite computacional del modelo de Turing, pues el lenguaje L_d no puede ser reconocido por ninguna máquina de Turing y en consecuencia L_d es un lenguaje no recursivo enumerable. En el teorema 2.2 se estableció que el universo de lenguajes sobre un alfabeto finito no puede ser numerado por un método universal que considere a todos sus elementos. Dado que un problema puede ser representado por medio de un lenguaje, lo anterior sugiere que deben existir más problemas que no podremos analizar con el mecanismo de Turing, de modo que los lenguajes recursivos enumerables se convierten en los únicos conjuntos que podemos analizar mediante el dispositivo abstracto de Turing.

Esta limitación es conocida como la tesis de Turing-Church y aunque es una interpretación entre varias que existen, en términos prácticos responde a las preguntas ¿Qué puede ser calculado? y ¿Qué no puede ser calculado? Aunque no se ha demostrado si es verdadera o falsa sirve como punto de partida para "formalizar la noción de algoritmo adoptando a las máquinas de Turing que se detienen para toda entrada como una noción formal que precisa de la idea intuitiva de un algoritmo" [42].

Entre los diversos formalismos propuestos para formalizar la noción de "procedimiento efectivo" propuesto por David Hilbert (véase, [2], [1]) encontramos los sistemas de Post (Emilio Post, [5], [7]), las funciones μ-recursivas (Kurt Godel, [3]), el λ-cálculo (Alonzo Church, [4]) y diversas variantes del modelo estándar de Turing con los cuales se llega a las mismas conclusiones de que no existe un mecanismo más general de cálculo que la máquina de Turing [40]. En el siguiente capítulo hablaremos de las variantes de máquinas de Turing que aumentan la "velocidad" de cómputo y que definen la noción de complejidad computacional.

Capítulo 3

Complejidad computacional

"No podemos volver atrás, por eso cuesta elegir.
Hay que tomar la decisión correcta. Mientras no
elijas, todo sigue siendo posible."

Nemo Nobody.

3.1. Modificaciones del modelo de Turing

Al final del capitulo 2 se habló brevemente de la famosa tesis de Turing-Church que establece en términos prácticos que no existe un mecanismo más universal que la máquina de Turing. Los diferentes modelos de cómputo existentes son el resultado de la búsqueda de un dispositivo u otros formalismos con más capacidad para calcular.

Aunque existen una cantidad considerable de variantes del modelo estándar presentado en 2.12, todos estos modelos son equivalentes entre sí. Nosotros nos enfocaremos en presentar las variantes del modelo estándar que consideramos indispensables para definir las nociones de complejidad computacional haciendo caso omiso de la gran mayoría de ellos.

3.1.1. Modelo de múltiples cintas

El modelo de Turing de múltiples cintas o modelo de k-cintas es una especie de extensión del modelo estándar, que consta al igual que éste último de una unidad de control conformada por un conjunto finito de estados y de varias cintas independientes, todas ellas con sus respectivas cabezas de lectura/escritura que se extienden de derecha a izquierda indefinidamente.

Inicialmente una máquina de múltiples cintas tiene escrita la cadena de entrada en la primera cinta y en las cintas restantes contiene espacios en blanco. Las posiciones de las cabezas pueden encontrarse dispersas aleatoriamente a lo largo de las cintas o en la primera casilla de cada cinta (véase Figura 3.1).

A cada cinta de un modelo múlti-cintas le corresponde una configuración, es decir, Si una máquina de múltiples cintas consta de un número finito de k cintas, entonces existen k configuraciones. De manera que por cada transición realizada por la máquina hay k configuraciones distintas por cada cinta que conforma a la máquina.

Una computación de una máquina de múltiples cintas consiste de las siguientes acciones:

1. El cambio de estado en el control finito.
2. Escribir un nuevo símbolo sobre algunas o todas las celdas con las cabezas en las cintas.
3. Mover en un movimiento algunos o todas las cabezas, independientemente a la izquierda, derecha o mantenerlos estacionarios en su posición.

Formalmente, denotamos a una máquina de Turing de k-cintas con el modelo $(Q,\Sigma,\Gamma,\delta,q_0,F)$ tal como de definió en 2.12 donde todos los componentes tienen el mismo significado excepto la función de transición que se define por,

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{R,L,S\}^k$$

La constante k representa el número de cintas que conforman la máquina con $k \geq 2$ y sus transiciones se escriben por $\delta(q_i,(\sigma_1,\sigma_2,\ldots,\sigma_k)) = (q_j,(\gamma_1,\gamma_2,\ldots,\gamma_k),(D_1,D_2,\ldots,D_k))$, donde q_i representa al estado "actual" y $\sigma_1,\sigma_2,\ldots,\sigma_k$ son los símbolos escaneados sobre las cintas $1,2,\ldots,k$ respectivamente. Los símbolos $\gamma_1,\gamma_2,\ldots,\gamma_k$ reescriben a los símbolos leídos sobre cada cinta de la máquina, $D_i \in \{R,L,S\}$ para cada $i \in \{1,2,3,\ldots,k\}$ y q_j representa al "nuevo estado".

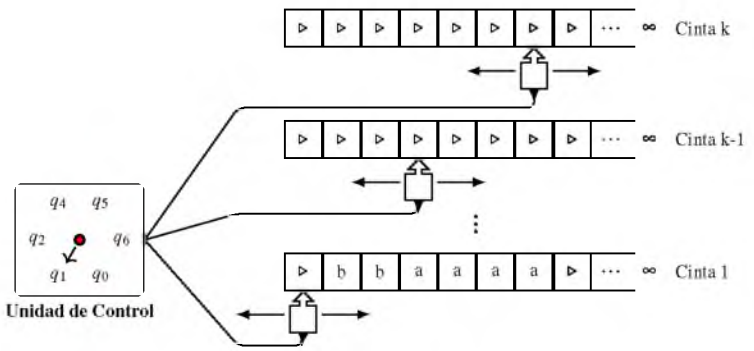


Figura 3.1: Máquina de múltiples cintas.

Como ejemplo, podemos pensar en el diseño de una máquina con dos cintas que reconozca al lenguaje formado de cadenas palindromos, ésto es $L_{pal} = \{\omega \in \Sigma^* \mid \omega = \omega^R\}$ donde $\Sigma = \{a,b\}$. Los siguientes pasos describen el funcionamiento de este mecanismo:

1. En la cinta 1 se escribe la cadena de entrada ω delimitada por dos espacios en blanco; uno en la primera celda de la cinta y el otro al final de la palabra. Las cabezas se posicionan en la primera celda de cada cinta.

2. Se realiza la copia de cadena de entrada ω en la segunda cinta, al finalizar éste paso del procedimiento, el primer cabezal queda posicionado en la celda vacía al final de la cadena y en la misma dirección, se posiciona el segundo cabezal. Después el cabezal regresa a la primera celda vacía al principio de la cadena.

3. El cabezal de la primera cinta se mueve hacia la celda vacía más a la derecha, mientras el cabezal de la segunda cinta se mueve hacia la primera celda vacía más a la izquierda. Durante este proceso ambas cabezas comparan los símbolos de cada celda, y en caso de que los símbolos sean diferentes en algún momento de la comparación la máquina se detiene y la cadena en consecuencia no pertenecerá a L_{pal}. En caso contrario si las cabezas llegan a su destino final la cadena es aceptada.

El modelo $M_{pal} = (\{q_0,q_1,q_2,q_3,q_4\},\{a,b\},\{a,b,\triangleright\},\delta,\{q_0\},\{q_4\})$ corresponde a la máquina de Turing de dos cintas del procedimiento anterior. La función de transición de M_{pal} se define bajo las siguientes reglas:

$$\begin{array}{lcllcl}
\delta(q_0,(\triangleright,\triangleright)) & = & (q_1,(\triangleright,\triangleright),(R,R)) & \delta(q_2,(\triangleright,b)) & = & (q_2,(\triangleright,b),(S,L)) \\
\delta(q_1,(a,\triangleright)) & = & (q_1,(a,a),(R,R)) & \delta(q_2,(\triangleright,\triangleright)) & = & (q_3,(\triangleright,\triangleright),(L,R)) \\
\delta(q_1,(b,\triangleright)) & = & (q_1,(b,b),(R,R)) & \delta(q_3,(a,a)) & = & (q_3,(a,a),(L,R)) \\
\delta(q_1,(\triangleright,\triangleright)) & = & (q_1,(\triangleright,\triangleright),(S,L)) & \delta(q_3,(b,b)) & = & (q_3,(b,b),(L,R)) \\
\delta(q_2,(\triangleright,a)) & = & (q_2,(\triangleright,a),(S,L)) & \delta(q_3,(\triangleright,\triangleright)) & = & (q_4,(\triangleright,\triangleright),(R,R))
\end{array}$$

El diagrama de transición que describe a una máquina de Turing k-cintas es similar a los diagramas usados en los modelos de las secciones anteriores. En una instrucción de la forma $\triangleright\triangleright \rightarrow aa,RR$, la expresión $\triangleright\triangleright$ significa que el primer espacio en blanco es el símbolo leído por el cabezal de la cinta 1 y el segundo espacio en blanco indica el símbolo leído en la cinta 2. La expresión aa remplaza a los símbolos leídos sobre las cintas y finalmente RR indica el movimiento de la primera y segunda cinta respectivamente. El diagrama 3.2 corresponde a la máquina de múltiples cintas definida con las reglas anteriores que acepta el lenguaje de palindromos sobre Σ.

3.1.2. Modelo no determinista

El modelo de Turig presentado en la definición 2.12 es conocido como modelo determinista, debido a que las computaciones del mecanismo se realizan de forma secuencial; esto se puede apreciar en los ejemplos correspondientes a la sección 2.4, donde observamos que la aceptación de una cadena resulta exitosa cuando la computación de una máquina llega a un estado final.

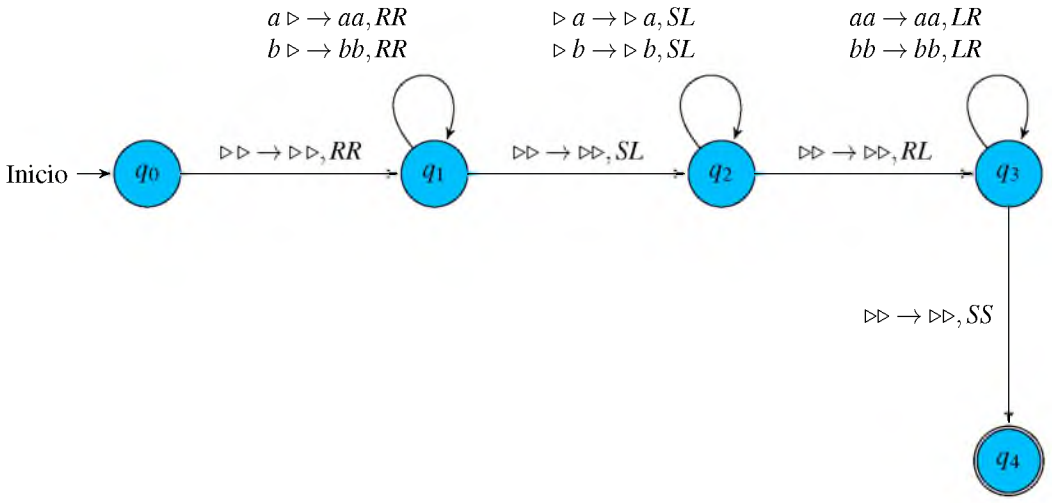


Figura 3.2: Máquina de múltiples cintas M_{pal}.

La máquina de Turing no determinista contiene los mismos elementos del modelo estándar, con la excepción de que durante el procesamiento de una cadena puede existir más de una configuración por cada transición aplicada. El modelo no determinista se representa por $(Q, \Sigma, \Gamma, \delta, q_0, F)$, donde todos los componentes tienen el mismo significado de la definición 2.12 con excepción de la función de transición que escribe por:

$$\delta : Q \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{R, L, S\})$$

Una manera de interpretar el funcionamiento de una máquina de Turing no determinista es pensar que sus computaciones se realizan en paralelo describiendo una especie de árbol donde sus ramas están compuestas de múltiples secuencias de configuraciones. Del árbol de computaciones podemos obtener los siguientes casos:

- Existe una o más trayectorias de aceptación, formada por secuencias de configuraciones $C_0, C_1, C_2, \ldots, C_n$, donde el estado de la configuración C_n pertenece al conjunto de estados finales y la máquina se detiene.
- Existen una o más trayectorias de no aceptación, donde el estado de la configuración final C_n no pertenece al conjunto de estados finales y la máquina se detiene.
- Existen una o más trayectorias que llevan a un bucle infinito, donde la máquina no se detiene.

En la figura 3.3 se ilustran las ideas descritas en los casos anteriores. Con color rojo los casos de rechazo y con azul los casos de aceptación, así como las trayectorias que llevan a un ciclo.

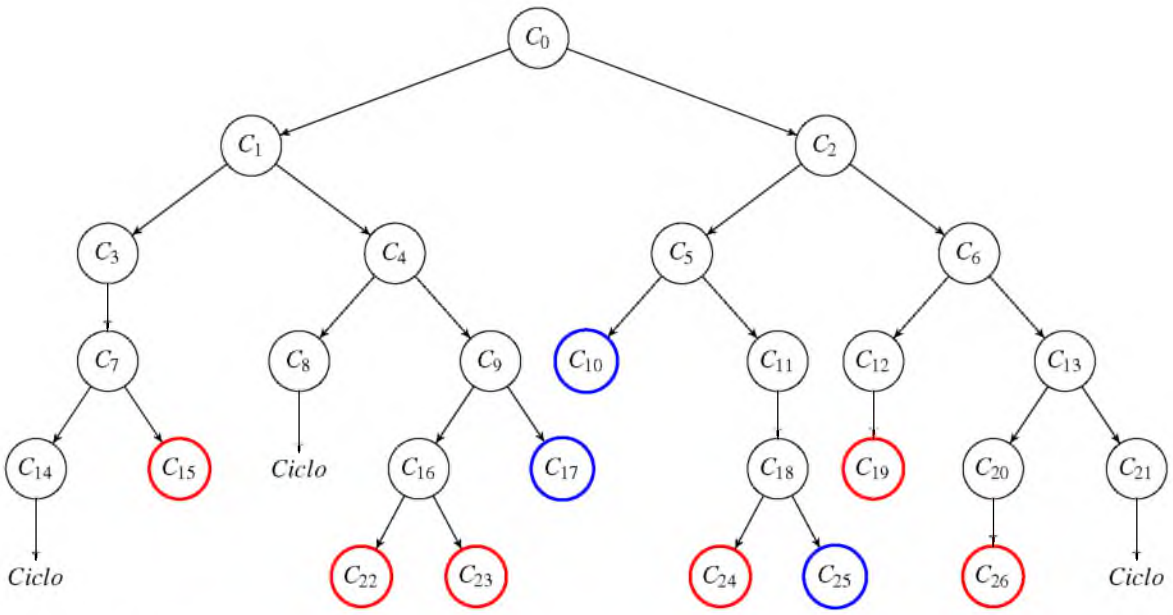


Figura 3.3: Árbol de computación hipotético de una máquina de Turing no determinista.

Definición 3.1. *Sea M_N una máquina de Turing no determinista y sea ω una cadena sobre un alfabeto finito. La cadena ω es aceptada por M_N si y sólo si existe una computación de M_N sobre ω que se detiene en un estado final. Una computación se denota por $\tau_{M_N}(\omega)$ y su tamaño está dado por la longitud de la trayectoria de aceptación más corta.*

La existencia de por lo menos una trayectoria de aceptación en el árbol de una máquina de Turing no determinista M_N sobre una cadena ω, es suficiente para establecer que la máquina acepta a la cadena ω. En caso contrario, es decir si no existe ninguna trayectoria de aceptación o existen solo las trayectorias con ciclos, la máquina M rechaza la cadena ω.

Definición 3.2. *Una máquina de Turing no determinista M_N reconoce a un lenguaje $L \subset \Sigma^*$, si M acepta a toda cadena $\omega \in L$ y rechaza a toda cadena $\omega \in \Sigma^* - L$, el lenguaje de la máquina M_N se denotará por $L(M_N) = \{\omega \mid M_N \text{ acepta a } \omega\}$.*

Veamos un ejemplo para comprender la noción no determinista, consideremos a todas las cadenas sobre el alfabeto finito $\Sigma = \{a,b\}$ que contienen la letra b en la antepenúltima posición. Nosotros podemos definir una máquina de Turing no determinista que reconozca a este lenguaje por medio del modelo $M_{Nb} = (\{q_0,q_1,q_2,q_3,q_4\},\{a,b\},\{\triangleright,a,b\},\delta,\{q_0\},\{q_4\})$, la función de transición δ está definida en la tabla 3.1.

δ	$\triangleright$	a	b
q_0	$(q_1, \triangleright, R)$	–	–
q_1	–	(q_1, a, R)	(q_1, b, R) (q_2, b, R)
q_2	–	(q_3, a, R)	(q_3, b, R)
q_3	–	(q_4, a, R)	(q_4, b, R)
q_4	–	(q_0, a, R)	(q_0, b, R)

Tabla 3.1: Función de transición de M_{Nb}.

El diagrama de transición para éste modelo no determinista se muestra en la figura 3.4, se puede notar claramente que en el nodo q_1 existen dos posibles cambios de estados ante la lectura del símbolo b, ésta simple ramificación conlleva a múltiples computaciones.

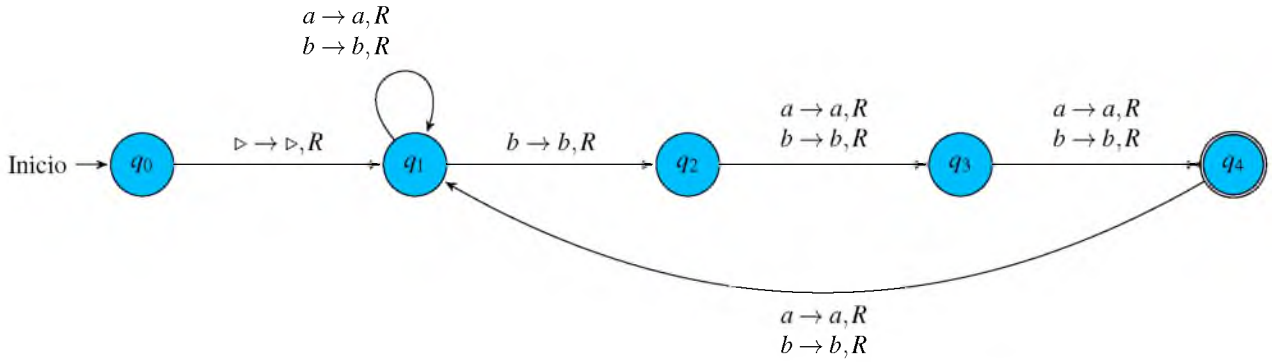


Figura 3.4: Diagrama de transición para M_{Nb}.

En la figura 3.5 muestra el árbol de computaciones de la máquina M_{Nb} sobre la cadena de entrada *abababab*. La trayectoria remarcada sobre el árbol de computación nos indica la ruta de aceptación de la cadena que llega a una configuración final que contiene al estado $q_3 \in F$. Cualquier otra trayectoria conduce a la detención de la máquina sin reconocer la cadena.

3.1.3. Modelo no determinista de varias cintas

El modelo de la máquina de Turing de múltiples cintas no determinista se representa por una 6-tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$ como en los modelos anteriores y con los mismos significados de los componentes de la definición 2.12 con la función de transición definida por:

$$\delta : Q \times \Gamma^k \to \mathscr{P}(Q \times \Gamma^k \times \{R, L, S\}^k)$$

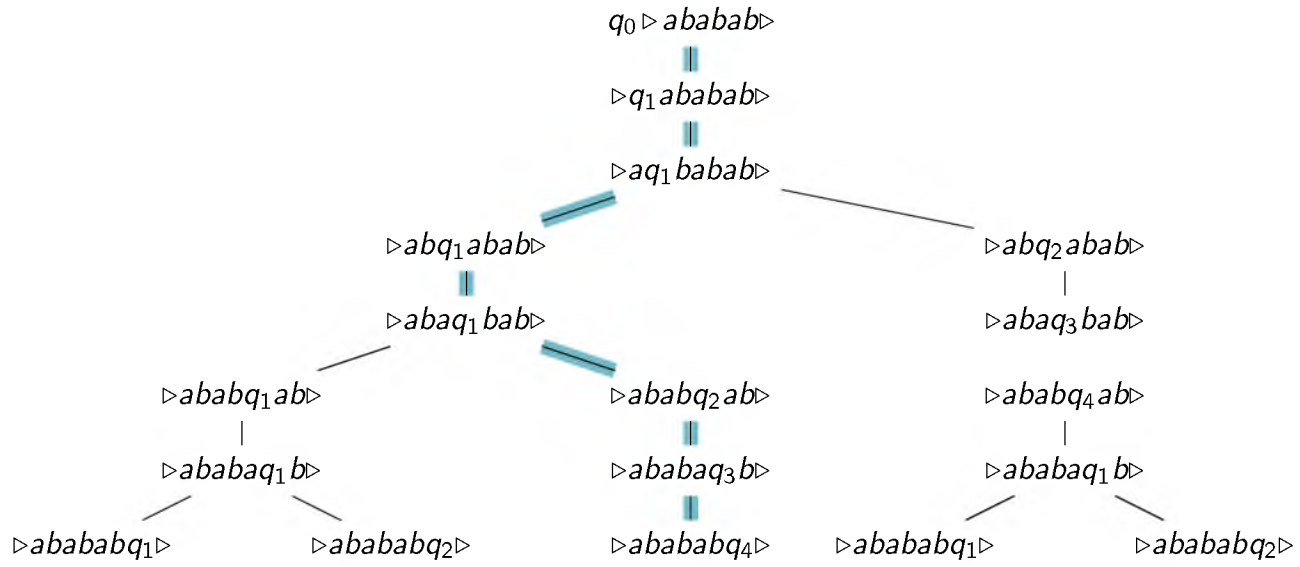


Figura 3.5: Árbol de computación de M_{nb} con la entrada *ababab*.

En el apartado anterior establecimos que una máquina de k-cintas genera k configuraciones al aplicar una transición, esto es, una configuración por cada cinta en un instante y en el modelo no determinista conformado por una cinta, la maquina genera múltiples configuraciones al aplicar las transiciones sobre una cadena de entrada. La máquina de Turing de k-cintas no determinista combina estas ideas, de manera que podemos obtener múltiples configuraciones por cada transición aplicada, para entender su funcionamiento consideremos el siguiente problema.

Problema general de partición: ¿Dado un conjunto de k números enteros no negativos $a_1, \ldots, a_k$, existe un subconjunto $I \subseteq \{1, \ldots, k\}$ tal que $\sum_{j \in I} a_j = \sum_{j \in I^c} a_j$?

Como primer paso necesitamos codificar de forma adecuada las instancias del problema, esto se puede realizar usando el alfabeto finito $\Sigma = \{a, b\}$, del cual las instancias del problema quedan representadas adecuadamente por las cadenas del siguiente lenguaje.

$$L_P = \{ab^{i_1}ab^{i_2}\cdots ab^{i_k} \mid i_1, \ldots, i_k > 0, \sum_{j \in I} i_j = \sum_{j \in I^c} i_j \text{ para algún } I \subseteq \{1, 2, \ldots, k\}\}$$

Este lenguaje consta de todas las cadenas sobre el alfabeto finito $\Sigma = \{a, b\}$ que representan la lista de enteros $i_1, \ldots, i_k$ que pueden ser partida en dos listas I e $\{1, 2, \ldots, k\}/I$ tal que la suma de los enteros de la lista I es igual a la suma de los enteros $\{1, 2, \ldots, k\}/I$. En particular, la cadena $ababbab \in L_P$ podemos escribirla como $ab^1ab^2ab^1$, obteniendo los valores de $i_1 = 1$, $i_2 = 2$ e $i_3 = 1$ del conjunto de tres elementos enteros $\{1, 2, 3\}$. Eligiendo la partición $I = \{1, 3\}$ obtenemos que,

$$\sum_{j \in I} i_j = i_1 + i_3 = 1 + 1 = \sum_{j \in I^c} i_j = 2.$$

El modelo $M_{Np} = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b\}, \{\triangleright, \#, a, b\}, \delta, \{q_0\}, \{q_6\})$ corresponde a la máquina no determinista de tres cintas que reconoce al lenguaje Lp con la función de transición δ ilustrada en el diagrama de la siguiente figura.

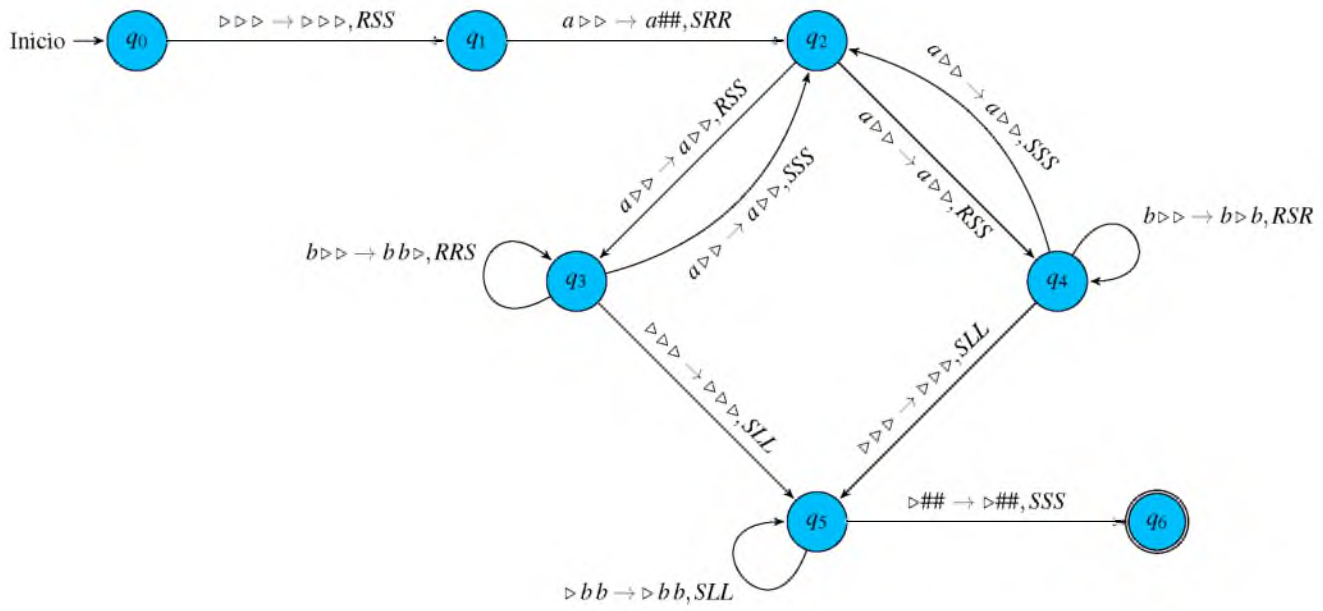


Figura 3.6: Diagrama de transición de M_{Np}.

El siguiente registro pertenece a la secuencia de aceptación de la cadena *ababbab* de la máquina M_{Np}, que inicia con los registros del lado izquierdo y finaliza en los registros del lado derecho. En ellos observamos que la máquina acepta una cadena cuando el número de letras $\mathtt{b}$ es igual en las cintas dos y tres.

$$
\begin{aligned}
q_0 \triangleright ababbab\triangleright,\ q_0\triangleright\triangleright\triangleright\triangleright,\ q_0\triangleright\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbq_4ab\triangleright,\ \#bq_4\triangleright\triangleright,\ \#bbq_4\triangleright\\
\vdash_M \triangleright q_1ababbab\triangleright,\ q_1\triangleright\triangleright\triangleright\triangleright,\ q_1\triangleright\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbq_2ab\triangleright,\ \#bq_2\triangleright\triangleright,\ \#bbq_2\triangleright\\
\vdash_M \triangleright q_2ababbab\triangleright,\ \#q_2\triangleright\triangleright\triangleright,\ \#q_2\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbaq_3b\triangleright,\ \#bq_3\triangleright\triangleright,\ \#bbq_3\triangleright\\
\vdash_M \triangleright aq_3babbab\triangleright,\ \#q_3\triangleright\triangleright\triangleright,\ \#q_3\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbabq_3\triangleright,\ \#bbq_3\triangleright,\ \#bbq_3\triangleright\\
\vdash_M \triangleright abq_3abbab\triangleright,\ \#bq_3\triangleright\triangleright,\ \#q_3\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbabq_5\triangleright,\ \#bq_5b\triangleright,\ \#bq_5b\triangleright\\
\vdash_M \triangleright abq_2abbab\triangleright,\ \#bq_2\triangleright\triangleright,\ \#q_2\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbabq_5\triangleright,\ \#q_5bb\triangleright,\ \#q_5bb\triangleright\\
\vdash_M \triangleright abaq_4bbab\triangleright,\ \#bq_4\triangleright\triangleright,\ \#q_4\triangleright\triangleright\triangleright &\quad \vdash_M \triangleright ababbabq_5\triangleright,\ q_5\#bb\triangleright,\ q_5\#bb\triangleright\\
\vdash_M \triangleright ababq_4bab\triangleright,\ \#bq_4\triangleright\triangleright,\ \#bq_4\triangleright\triangleright &\quad \vdash_M \triangleright ababbabq_6\triangleright,\ q_6\#bb\triangleright,\ q_6\#bb\triangleright
\end{aligned}
$$

3.2. Medidas de tiempo computacional

En los artículos de Michael O. Rabin [8], J. Hartmanis y R. E. Stearns [9] y de Michael Blum [11] se desarrolló la noción de medida de complejidad de un problema en términos del número de pasos requeridos para resolverlo [30]. En especial, el trabajo de J. Hartmanis y R. E. Stearns describe que el cálculo realizado por un modelo de k-cintas sobre una cadena de entrada es más eficiente que el cálculo realizado por una máquina de Turing estándar, cuando ambos mecanismos reconocen al mismo lenguaje.

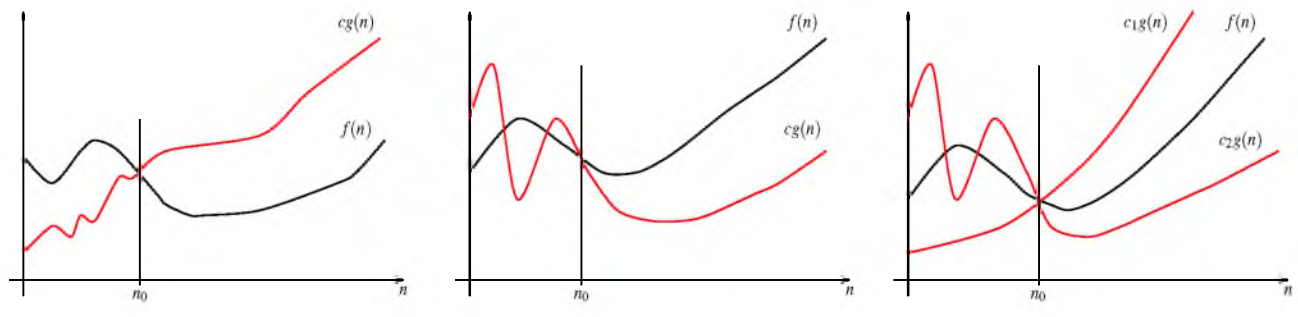
El tiempo de computación de una máquina de Turing estándar o de sus variantes es medido por el número de transiciones aplicadas durante el procesamiento de una cadena de entrada y se describe mediante una función de crecimiento sobre los números naturales. Las siguientes definiciones sobre las medidas de complejidad son estándar y se requieren para definir las clases de problemas tratables e intratables [60].

3.2.1. Funciones de crecimiento

Definición 3.3. *Para una función $g : \mathbb{N} \to \mathbb{N}$, tenemos los siguientes conjuntos que denotan las medidas de complejidad temporal también llamadas orden de complejidad,*

- $O(g(n)) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists c > 0 \text{ y } n_0 \in N, 0 < f(n) \leq cg(n), \forall n \geq n_0\}$, *denota al conjunto de funciones acotadas superiormente por la función g, donde $c \in \mathbb{R}$,*
- $\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists c > 0, n_0 \in N, 0 < cg(n) \leq f(n), \forall n \geq n_0\}$, *denota al conjunto de funciones acotadas interiormente por la función g, donde $c \in \mathbb{R}$,*
- $\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{N} \mid \exists c_1, c_2 > 0, n_0 \in N : 0 < c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$, *donde $c \in \mathbb{R}$.*

Las notaciones anteriores son usadas para describir el peor caso del tiempo de ejecución de una máquina de Turing, las gráficas siguientes muestran las interpretaciones de los conjuntos presentados en la definición anterior.



(a) $f(n) \in O(g(n))$ (b) $f(n) \in \Omega(g(n))$ (c) $f(n) \in \Theta(g(n))$

Figura 3.7: Funciones asintóticas de crecimiento.

La construcción de la función $f : \mathbb{N} \to \mathbb{N}$ se lleva a cabo realizando una relación entre el tamaño de la cadena de entrada y el número de transiciones procesadas durante su computación sobre la máquina de Turing, la siguiente definición formaliza esta idea para una máquina de Turing determinista, no obstante, la definición puede ser extendida al modelo de varias cintas o a cualquier otra variante del tipo determinista.

3.2.2. Tiempo computacional del modelo determinista

Definición 3.4. *Sea M_D una máquina de Turing determinista. El tiempo de complejidad de M_D, es una función $T_{M_D} : \mathbb{N} \to \mathbb{N}$, tal que $T_{M_D}(n)$ al número máximo de transiciones procesadas por una computación de M_D cuando ha sido iniciada con una cadena ω de tamaño n, ésto es,*

$$T_{M_D}(n) = máx\{\tau_{M_D}(\omega) \mid \ donde\ l(\omega) \leq n\}.$$

Retomemos al lenguaje $L_{pal} = \{\omega \in \Sigma^* \mid \omega = \omega^R\}$ sobre $\Sigma = \{a,b\}$, el diseño de una máquina de Turing estándar para L_{pal} se define por $M_{pal_1} = (\{q_0,q_1,\ldots,q_7\},\{a,b\},\{\triangleright,a,b\},\delta,q_0,\{q_7\})$ con la función δ mostrada en el diagrama de transición de la siguiente figura.

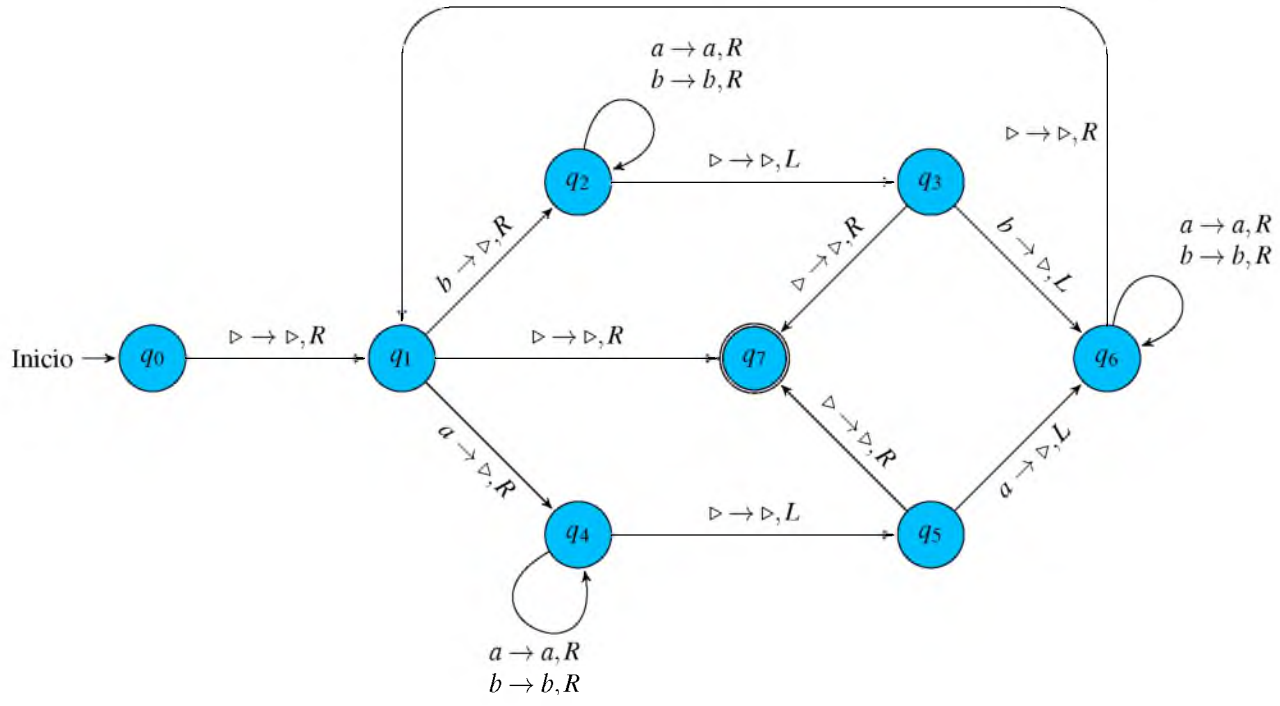


Figura 3.8: Diagrama de transición M_{pal} estándar.

Encontrar la función explicita que describe el tiempo de complejidad de una máquina de Turing no es tarea simple, ya que no existe un método universal para hacerlo. En la tabla 3.2 se puede apreciar el número de transiciones procesadas al ejecutar el mecanismo anterior sobre las cadenas de tamaño 1, 2, 3 respectivamente. De esos cálculos nos percatamos que los máximos de cada caso son $T_{M_{pal_1}}(0) = 2$, $T_{M_{pal_1}}(1) = 4$, $T_{M_{pal_1}}(2) = 7$ y $T_{M_{pal_1}}(3) = 11$.

A partir de la obtención de los máximos se requiere de un detallado análisis de las computaciones generadas por la máquina M_{pal}. De la figura 3.8 se puede observar que la ejecución de una cadena sobre M_{pal_1} de tamaño n par que comience con el símbolo a, realiza un recorrido de q_1 a q_4 procesando $n+1$ transiciones.

Del estado q_5 al q_6 se procesan n transiciones completándose un ciclo, obteniendo una cadena de longitud $n-2$. Al aplicar el procedimiento descrito anteriormente, y al ser similar el recorrido para eliminar el símbolo a y b del palindromo se puede obtener la secuencia $n-1, n-2, \ldots, 1, 1$ de registros al procesar la cadena par completa. Sumando las computaciones observamos que ésta corresponde a la suma de los primeros $n+1$ números naturales más uno. Análogamente se obtiene el mismo resultado con los palindromos de longitud impar, obteniéndose así la función,

$$T_{M_{pal_1}}(n) = \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} + 1.$$

Aplicando la definición $O(g(n))$ a la función $T_{M_{pal_1}(n)}$ vemos que para un $c = 3$ y $n_0 = 1$, la función $\mathtt{t}_{M_{pal_1}(n)}$ se encuentra acotada superiormente por la función n^2 y en consecuencia la función $T_{M_{pal_1}(n)}$ se dice que es de orden polinomial, ésto es, $T_{M_{pal_1}(n)} \in O(n^2)$.

Tamaño 0	Tamaño 1	Tamaño 2	Tamaño 3
$\tau_{M_{pal}}(\triangleright\triangleright) = 02$	$\tau_{M_{pal}}(\triangleright\, a\, \triangleright) = 04$	$\tau_{M_{pal}}(\triangleright\, a\, a\, \triangleright) = 07$	$\tau_{M_{pal}}(\triangleright\, a\, a\, a\, \triangleright) = 11$
	$\tau_{M_{pal}}(\triangleright\, b\, \triangleright) = 04$	$\tau_{M_{pal}}(\triangleright\, a\, b\, \triangleright) = 05$	$\tau_{M_{pal}}(\triangleright\, a\, a\, b\, \triangleright) = 06$
		$\tau_{M_{pal}}(\triangleright\, b\, a\, \triangleright) = 05$	$\tau_{M_{pal}}(\triangleright\, a\, b\, a\, \triangleright) = 11$
		$\tau_{M_{pal}}(\triangleright\, b\, b\, \triangleright) = 07$	$\tau_{M_{pal}}(\triangleright\, a\, b\, b\, \triangleright) = 06$
			$\tau_{M_{pal}}(\triangleright\, b\, a\, a\, \triangleright) = 06$
			$\tau_{M_{pal}}(\triangleright\, b\, a\, b\, \triangleright) = 11$
			$\tau_{M_{pal}}(\triangleright\, b\, b\, a\, \triangleright) = 06$
			$\tau_{M_{pal}}(\triangleright\, b\, b\, b\, \triangleright) = 11$

Tabla 3.2: Computación de M_{pal} estándar.

En contraste con el modelo determinista de una cinta para reconocer al lenguaje L_{pal}, la máquina de múltiples cintas definida de la figura 3.2 tiene un tiempo de operación mucho menor al del modelo anterior, los siguientes registros pertenecen a la cadena de entrada *aabaa* sobre el mecanismo de múltiples cintas la figura 3.2.

$q_0BaabaaB, q_0BBBBBBB$	$Baabaaq_2B, Baabaq_2aB$	$Baabaq_3aB, Bq_3aabaaB$
$Bq_1aabaaB, Bq_1BBBBBB$	$Baabaaq_2B, Baabq_2aaB$	$Baabq_3aaB, Baaq_3abaaB$
Baq_1abaaB, Baq_1BBBBB	$Baabaaq_2B, Baaq_2baaB$	$Baaq_3baaB, Baaq_3baaB$
$Baaq_1baaB, Baaq_1BBBB$	$Baabaaq_2B, Baq_2abaaB$	$Baq_3abaaB, Baabq_3aaB$
$Baabq_1aaB, Baabq_1BBB$	$Baabaaq_2B, Bq_2aabaaB$	$Bq_3aabaaB, Baabaq_3aB$
$Baabaq_1aB, Baabaq_1BB$	$Baabaaq_2B, q_2BaabaaB$	$q_3BaabaaB, Baabaaq_3B$
$Baabaaq_1B, Baabaaq_1B$		$q_4BaabaaB, Baabaaq_4B$

Los registros de la izquierda pertenecen a la primera parte del procedimiento de aceptación, de los cuales observamos que se generan seis registros a partir de la configuración inicial. Los registros del centro pertenecen a la segunda parte del proceso, donde se generan seis registros más y finalmente los siete registros de la derecha pertenecen al proceso final.

Si observamos detalladamente el proceso de aceptación de una cadena de tamaño n, en la primera etapa se generan $n+1$ registros, en la segunda etapa $n+1$ registros más y finalmente en la tercera etapa $(n+1)+1$ registros, sumando todos estos registros obtenemos la función $T_{M_{pal}}(n) = 3(n+1)+1$, la cual es de orden $O(n)$. Esto quiere decir que un modelo de Turing de múltiples cintas es más "veloz" que el modelo de Turing estándar.

3.2.3. Tiempo computacional del modelo no determinista

Pensar en lo que significa tiempo de complejidad de una máquina de Turing no determinista podría generar ciertas complicaciones debido a la naturaleza del mecanismo, sin embargo, recordemos que la longitud de una computación no determinista es la secuencia de aceptación más corta que termina en un estado final.

Definición 3.5. *Sea M_N una máquina de Turing no determinista y sea ω una cadena sobre un alfabeto finito. El tiempo de complejidad de M_N es la función $T_{M_N(n)} : \mathbb{N} \to \mathbb{N}$ tal que $T_{M_N}(n)$ es el máximo número de transiciones procesadas por una computación que termine en un estado de aceptación, expresado por*

$$T_{M_N}(n) = max\{\tau_M(w) \mid \omega \in \Sigma^*, l(\omega) \leq n \text{ y } M \text{ acepta } a\ w\}.$$

Es importante enfatizar que el tamaño de la computación en una máquina no determinista es la secuencia de configuraciones más corta que termina en un estado final. El proceso para encontrar la función $T_{M_N}(n)$ es similar al proceso sugerido en el caso determinista.

Tomando como ejemplo la máquina de la figura 3.4, en el mejor de los casos este mecanismo puede estacionarse en el estado q_1 hasta elegir el cambio hacia el estado q_2 justo en la antepenúltima letra b, logrando que cualquier cadena de tamaño n tenga una computación de longitud $n+2$ y en consecuencia la función $T_{M_{nb}}(n) \in O(n)$.

Otro ejemplo de complejidad no determinista se obtiene de la máquina de la figura 3.6. Su función de complejidad no es difícil de calcular, siguiendo un procedimiento parecido al presentado en la tabla 3.2, se buscan las secuencias de aceptación de menor longitud en cada caso y se analiza el historial de cómputo como en el caso determinista. De esa forma obtenemos que el orden de complejidad computacional para la máquina M_{Np} de la figura 3.6 es $O(n^2)$.

El problema general de partición en su versión para máquina de Turing esta fuertemente ligado al problema que se expondrá en la sección 3.4. En particular el problema de partición pertenece a la clase de problemas intractables. Los resultados de la siguiente sección nos ayudarán a reforzar la noción de problemas tractables e intractables.

3.3. Relación de complejidad entre modelos

Los siguientes resultados fueron tomados del libro de Michael Sipser [51].

Teorema 3.1. *Si M es máquina de Turing de múltiples cintas que ejecuta una cadena de entrada ω en $t(n)$ movimientos con $t(n) > n$, entonces existe una máquina de Turing estándar S equivalente que con la misma entrada ω su tiempo de complejidad es de $O(n^2)$.*

Demostración. Dada una máquina de Turing de múltiples cintas $M = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_0, F_1)$, debemos construir una máquina estándar $S = (Q_2, \Sigma_2, \Gamma_2, \delta_2, p_0, F_2)$ que simule las transiciones de la máquina M. Para ello tendremos que representar la información de cada cinta de la máquina M junto con las respectivas posiciones en que se encuentran cada una de sus cabezas y escribirlas sobre la cinta de la máquina estándar S.

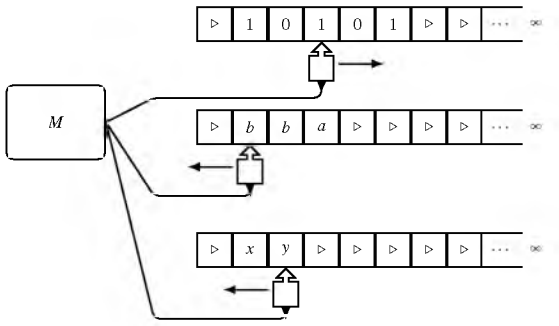


Figura 3.9: Configuración de la máquina M.

La idea básica para establecer lo anterior consiste en agregar el símbolo # y todos los elementos del alfabeto Γ_1 al alfabeto Γ_2 de la máquina S. El símbolo # se usará como delimitador para separar la información de las diferentes cintas de la máquina M que serán escritas sobre la cinta de la máquina S, además este carácter servirá para indicar el principio y fin de toda la información codificada en la cinta de la máquina S. Durante la simulación, la máquina S ubicará las cabezas de las diferentes cintas de la máquina M subrayando el símbolo al que apuntan las cabezas de M en un instante. Por lo tanto, para cada $\sigma \in \Gamma_1$ se agregará el símbolo $\underline{\sigma}$ al alfabeto Γ_2.

En la figura 3.9, se observa una configuración hipotética de la máquina de Turing M de varias cintas previo a la aplicación de una transición y en la figura 3.10 observamos la máquina de Turing S estándar que simula la transición de M. La simulación se realiza ralentizando el proceso de la transición de M, es decir, la máquina S como primer paso recorre la cinta de principio a fin, durante este paso la máquina S almacena en la unidad de control la información de los símbolos marcados que indican las posiciones de los cabezas de la máquina M. Una vez hecho lo anterior, el cabezal regresa a la posición inicial colocándose en el primer símbolo #.

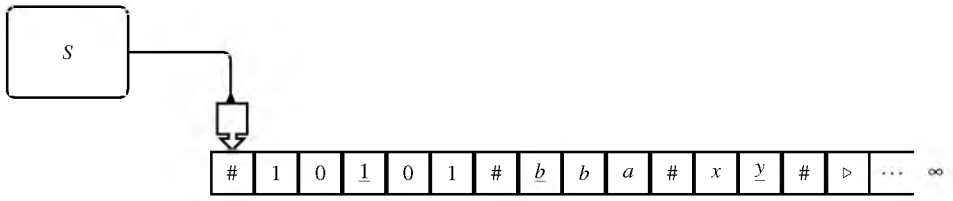


Figura 3.10: Codificación en Máquina de Turing S.

Como segundo paso la máquina S realiza nuevamente el recorrido de principio a fin, pero esta vez sustituye los símbolos remarcados por los nuevos símbolos que se remplazarán en las cintas de la máquina M, por ejemplo supongamos que la configuración hipotética de la figura 3.9 tiene por transición $\delta(q_i,(1,b,y)) = (q_j,(0,a,x),(R,L,L))$, para $q_i, q_j \in Q$. En este caso durante el recorrido se actualizarán los símbolos marcados con la información de la transición y el cabezal de la máquina S regresará nuevamente a la posición del primer símbolo # una vez más.

Finalmente la máquina S realiza un último recorrido de principio a fin con la intención de actualizar los símbolos marcados en su cinta que simulan las posiciones de las cabezas en la máquina M. Al realizar lo anterior el cabezal de la máquina S regresa a la posición inicial tal como se observa en la siguiente figura.

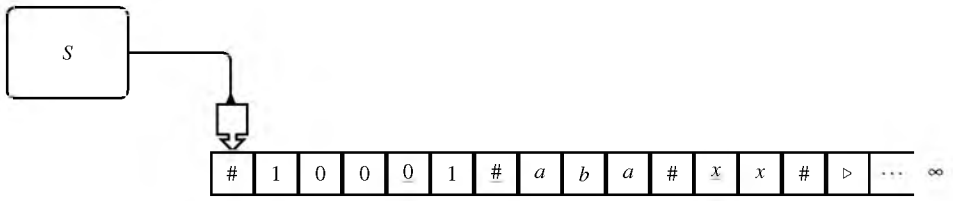


Figura 3.11: Actualización final en máquina S.

En la figura 3.11 observamos un símbolo $\underline{\#}$, como describimos anteriormente el primer paso de la simulación consiste en identificar a los símbolos marcados sobre la cinta de S, durante el primer paso del recorrido si el cabezal de la máquina S llega al símbolo marcado $\underline{\#}$, entonces la máquina realiza un desplazamiento de los caracteres restantes para colocar un símbolo en blanco marcado y sustituye el símbolo $\underline{\#}$ por un #, justo como se observa en la figura 3.12.

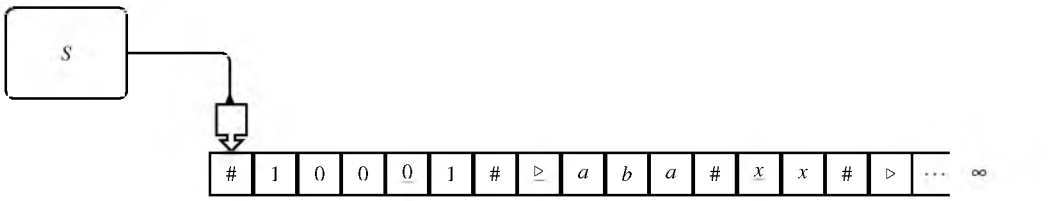


Figura 3.12: Recorrido en la cinta de S.

De los puntos anteriores observamos que la aplicación de una transición en la máquina M, requiere de tres recorridos en la máquina S, si una transición de la máquina M no esta definida, entonces la máquina S se detiene. De lo anterior se puede obtener el siguiente procedimiento algorítmico:

Algoritmo 2 (Simulación de máquina de K-cintas a modelo estándar)

1. Inicialización: Dada una entrada $\omega = \sigma_1\sigma_2\ldots\sigma_n$; la máquina S representa sobre su cinta la información de las k cintas de la máquina M como sigue:

$$\#\underline{\sigma_1}\sigma_2\sigma_3\ldots\sigma_n\#\underline{\triangleright}\#\underline{\triangleright}\#\ldots\#\underline{\triangleright}\#\triangleright\triangleright\triangleright\triangleright$$

2. Simulación de una transición: Para simular un movimiento de máquina M, la máquina S escanea su cinta desde el primer # al $(k+1)$-enésimo # final, para determinar los símbolos debajo de las cabezas virtuales. Luego la máquina S realiza otro recorrido para actualizar el contenido de la cinta de acuerdo con la forma que dicta la función de transición de la máquina M y finalmente realiza un último recorrido para actualizar las posiciones de las cabezas virtuales sobre su cinta.

3. Desplazamiento de contenido: Si durante la actualización, la máquina S mueve una de las cabezas virtuales a un #, el cual se convierte en $\underline{\#}$, eso significa que la máquina M necesita espacio para realizar su cálculo. En ese caso, la máquina S desplaza una casilla todo el contenido a la derecha entre el símbolo $\underline{\#}$ y el símbolo # final, para colocar un espacio en blanco y mover el cabezal virtual escribiendo $\underline{\triangleright}$.

4. Paso final: Si durante la simulación, la máquina M llega a un estado de aceptación, entonces la máquina S acepta y se detiene. Si la máquina M se detiene en un estado no final, la máquina S se detiene y rechaza. De lo contrario, se continúa la simulación M con el paso 2.

Para calcular el tiempo de complejidad de la simulación, supongamos que la máquina M realiza el procesamiento de una cadena de tamaño n en $t(n)$ pasos. La longitud del espacio ocupado en la cinta de la máquina S al codificar la cadena ω de tamaño n es de $3n+(k+1)$, donde $k+1$ es el número de símbolos # que divide la información de las k cintas de la máquina M. Observamos además que la aplicación de una transición de la máquina M en S requiere de $3(3n+(k+1))$ pasos. De la definición 3.3 no es difícil establecer que la función anterior se puede acotar en un tiempo $O(t(n))$.

Finalmente, la simulación total de todas las transiciones que conforman a la máquina M en la máquina S se realizan en $t(n) \times O(t(n)) = O(t^2(n))$. ∎

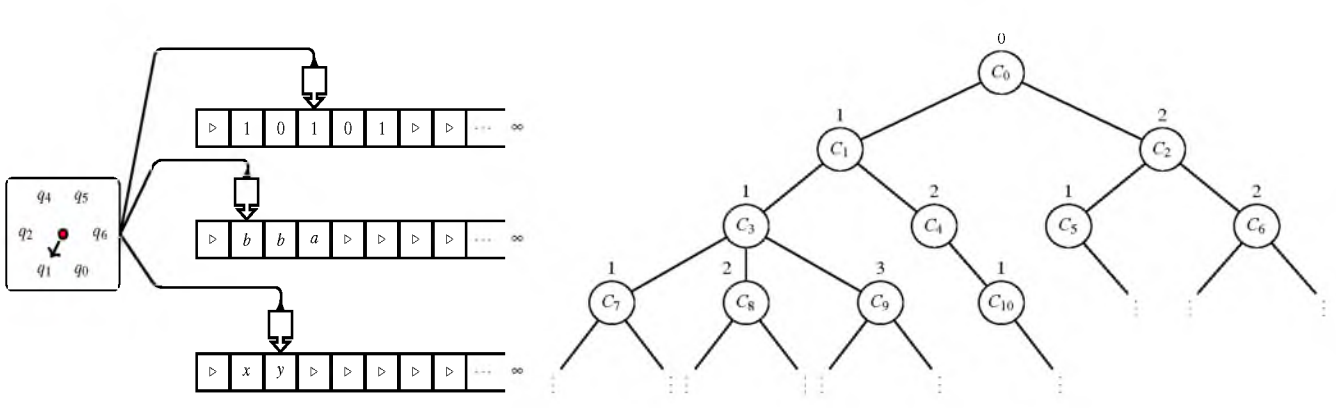
Con el teorema anterior no solo se demostró la relación de complejidad entre el modelo estándar y el de multiples cintas, sino que además se estableció que el modelo de varias cintas no es más general que el modelo estándar, algo parecido se puede apreciar en el siguiente teorema.

Teorema 3.2. *Si N es máquina de Turing no determinista que ejecuta una cadena de entrada ω en $t(n)$ movimientos con $t(n) > n$, entonces existe una máquina de Turing estándar S equivalente a la máquina N tal que para la misma entrada ω su tiempo de complejidad es $O(2^{t(n)})$.*

Demostración. Para demostrar el teorema debemos construir una máquina de Turing M de k-cintas que simule a la máquina no determinista N sobre cada entrada ω. Como hemos visto las computaciones de una máquina de Turing no determinista pueden visualizarse mediante un árbol donde cada nodo representa una configuración.

La idea básica de la simulación consiste en que la máquina M compruebe todos los caminos del árbol de configuraciones de la máquina N hasta encontrar una configuración que acepte la entrada ω. Si todas las ramas del cómputo de la máquina N son rechazadas, entonces la máquina M rechaza la cadena ω también, además si no hay ramas de aceptación en el cómputo de la máquina N y existe una rama que entra en un ciclo infinito, por consiguiente la máquina M también entrará en un ciclo un infinito.

La implementación se trabaja sobre una máquina M de tres cintas como la presentada en la figura 3.13(a), la primera cinta o "cinta de entrada" almacena la cadena de entrada ω, la cual nunca será alterada durante el proceso de la simulación, la segunda cinta es la denominada "cinta de simulación", porque almacena una copia de la información contenida en la cinta de la máquina N sobre alguna rama de su computación no determinista y finalmente la tercera cinta o "cinta de dirección" contendrá la localización de las configuraciones del árbol de computación de la máquina N.



(a) Máquina M que simula a N.

(b) Árbol de computación de N

Figura 3.13: Simulación de la máquina N sobre M.

Cada nodo del árbol de computaciones de la máquina N tendrá una dirección asociada representada por una cadena del alfabeto $\Sigma_b = \{1,2,\ldots,b\}$, esto suponiendo que cada nodo en el árbol de computación de la máquina N tiene a lo más b hijos, siendo b el entero que representa el número máximo de elecciones por cada transición de la máquina N. La forma en que los nodos serán etiquetados se observa en la figura 3.13(b).

La visita a los nodos se realiza por medio de una búsqueda a lo ancho que verifica cada nodo en orden lexicográfico, es decir, $\varepsilon, 1, 2, \ldots, b, 11, 12, \ldots, 1b, 21, 22, \ldots, 2b, \ldots$ y así sucesivamente. La cadena ε corresponde a la configuración inicial y si alguna cadena en el orden lexicográfico no tiene una secuencia en el árbol de computaciones de la máquina N, diremos que esa dirección no está definida.

El proceso de simulación se describe en el siguiente procedimiento:

Algoritmo 3 (Simulación de modelo no determinista a modelo de k-cintas)

1. Inicialización: La cinta de entrada de la máquina M contiene almacenada a la cadena ω y el resto de las cintas están vacías.

2. Copiado: Se realiza una copia del contenido de la cinta de entrada de la máquina M a la segunda cinta que simulará la computación de la máquina N.

3. Simulación: La segunda cinta es utilizada para simular a la máquina N con la entrada ω sobre una trayectoria de su árbol de computaciones. Antes de cada paso se consulta el símbolo próximo sobre la cinta 3 y se determina que elección realizar de las permitidas por la función de transición de la máquina N.

Si se alcanza una configuración de aceptación la máquina M se detiene y acepta. En caso de que se llegue a una configuración de rechazo, no hay más símbolos sobre la cinta tres que contiene la dirección de la rama simulada o la elección no determinista es invalida, se aborta la rama de la computación elegida y se continúa con el siguiente paso.

4. Actualización de la dirección: Se reemplaza la cadena en la cinta tres con una nueva en el orden establecido y se dirige al paso 2 para simulación de la nueva rama de la computación de N elegida.

Suponiendo que la máquina no determinista N tiene un tiempo computacional $t(n)$, observamos de la simulación anterior que una cadena de tamaño n sobre una rama de la computación elegida tiene a lo más una longitud de $t(n)$, dado que el número de elecciones por cada transición es a lo más b, entonces la búsqueda de todos los nodos en el árbol de computación tiene una cota $O(b^{t(n)})$. Dado que el tiempo de recorrido de la raíz o configuración inicial a cualquier nodo es $t(n)$, se sigue que el tiempo total de la máquina M es de $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

Del teorema 3.1 sabemos que la transformación de una máquina de k-cintas a una máquina estándar es de orden $O(n^2)$. De modo que el tiempo de simulación de la máquina no determinista N en una máquina estándar es $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$, demostrándose el teorema. ∎

3.4. Tractabilidad y la clase NP-Completa

Los lenguajes o problemas que se pueden resolver en tiempo polinómico con una máquina de Turing estándar se denominan "problemas tractables". Existen una infinidad de problemas de esta naturaleza, en este caso los ejemplos mostrados en el apartado 2.4.2 son lenguajes decidibles en tiempo polinómico.

Hemos visto que con una codificación apropiada para un determinado problema podemos construir una máquina de Turing adecuada que sea capaz de procesar las instancias del problema. Aunque los ejemplos a lo largo del capítulo son lenguajes simples, existen una gran variedad de problemas que pueden ser implementados mediante una máquina de Turing, un ejemplo sencillo al respecto es el algoritmo de la multiplicación que puede ser implementado en una máquina de Turing determinista de 3 cintas con orden polinómico usando cadenas sobre un alfabeto unario, esta máquina se presenta en la siguiente figura.

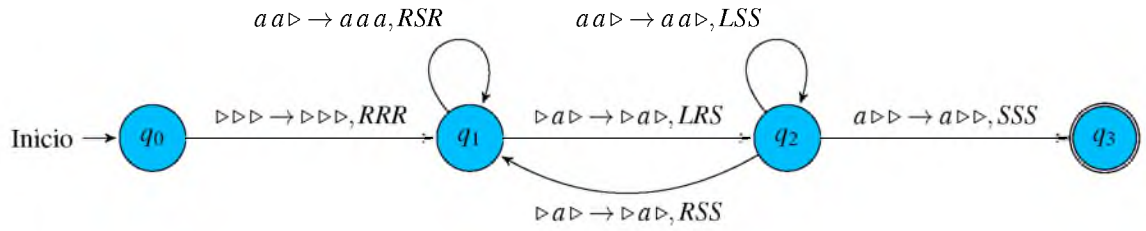


Figura 3.14: Máquina de Turing que multiplica.

Entre otros procedimientos de orden polinómico podemos encontrar: el algoritmo de Dijkstra con el que se puede calcular eficientemente la ruta más corta entre dos puntos de una gráfica[1], el algoritmo de patrones de emparejamiento, la transformada rápida de Fourier, el algoritmo para corrección de errores, entre otros. Los lenguajes o problemas con estas características forman parte de una muy importante familia de lenguajes que se define a continuación.

Definición 3.6. *Un lenguaje L es* ***decidible en tiempo polinomial*** *si existe una máquina de Turing M_D que reconozca a L en $\tau_M = O(n^r)$, donde $r \in \mathbb{N}$ independiente de n. Se denota a la familia de lenguajes decidibles en tiempo polinomial por*

$$\mathbf{P} = \bigcup_{r>1} \mathbf{DTIEMPO}(\mathbf{n^r}).$$

Por otro lado, los problemas que se deciden en una máquina de Turing estándar con orden exponencial reciben el nombre de "problemas intractables". En la figura 1.8 del capítulo 1 presentamos algunas funciones que podrían corresponder a problemas intractables. Los lenguajes que son reconocidos en tiempo polinomial en un modelo no determinista pertenecen a la siguiente familia de lenguajes.

[1] Termino relativo a la teoría de las gráficas.

Definición 3.7. *Un lenguaje L es* ***reconocible en tiempo polinomial no determinista*** *si existe una máquina de Turing M_N no determinista que reconoce a L en $\tau_M = O(n^r)$, donde $r \in \mathbb{N}$ independiente de n. Se denota a la familia de lenguajes reconocidos en tiempo polinomial no determinista por*

$$\mathbf{NP} = \bigcup_{r>1} \mathbf{NTIEMPO}(\mathbf{n^r}).$$

En 1971, el matemático Canadiense Steve Cook presentó un artículo en el que realizó un análisis del problema de satisfacibilidad de fórmulas booleanas y definió la clase de problemas **NP**-Completa que dió origen a la teoría de la complejidad computacional y la búsqueda de algoritmos eficientes para resolver problemas de esta clase [15].

Como hemos visto transformar una máquina de Turing no determinista a una máquina estándar requiere de un tiempo exponencial, de manera que cualquier lenguaje o problema reconocible en tiempo polinomial no determinista es intractable en un modelo determinista, sin embargo, la clase de lenguajes **NP**-Completa abre la posibilidad de resolver problemas **NP** de forma eficiente. Para comprender esto daremos una exposición del problema de Cook.

3.4.1. El problema de satisfacibilidad de fórmulas booleanas

De entrada es importante conocer un par de definiciones. En secciones pasadas observamos que las máquinas de Turing pueden ser vistas como reconocedoras de lenguajes o como mecanismos que resuelven problemas, una interpretación más es verlas como mecanismos que calculan funciones. La siguiente definición nos da una mejor idea sobre esto.

Definición 3.8. *Una función de cadena f es Turing-computable si existe una máquina de Turing determinista $M = (Q, \Sigma, \Gamma, q_0, F, \delta)$ tal que $q_0 w \vdash_M^* q_f u$ para algún $q_f \in F$, cuando $f(w) = u$.*

La definición anterior quiere decir que la función f transforma elementos de un lenguaje en elementos de otro, en este caso la máquina de Turing realiza ese proceso de transformación, un ejemplo de este tipo de mecanismo es la máquina de la figura de abajo que trasforma los símbolos a en b y viceversa de las cadenas del lenguaje sobre el alfabeto $\Sigma = \{a, b\}$, este proceso de transformación se conoce como reducción.

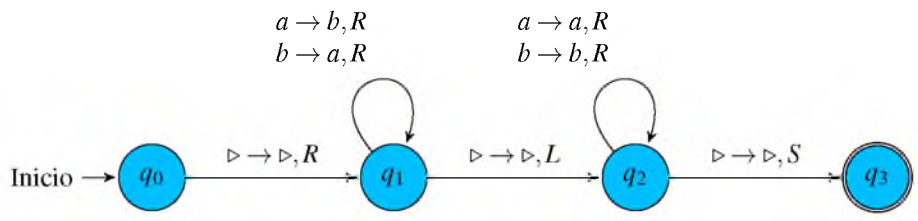


Figura 3.15: Máquina de Turing que complementa cadenas.

Definición 3.9. *Sean L y L' lenguajes sobre un alfabeto finito Σ. Decimos que L es Karp reducible en tiempo polinomial a L', denotado por $L \preceq_p L'$, si existe una función Turing-computable $f : \Sigma^* \to \Sigma^*$ tal que para cada $\omega \in \Sigma^*$, $\omega \in L$ si y solo sí $f(\omega) \in L'$.*

Teorema 3.3. *Sean L y L' lenguajes sobre un alfabeto finito Σ tal que $L \preceq_p L'$, entonces*

*a) Si $L' \in$ **P** entonces $L \in$ **P**.*

*b) Si $L' \in$ **NP** entonces $L \in$ **NP**.*

Demostración. a) Dado que $L' \in$ **P**, entonces existe una máquina de Turing determinista M'_D que decide si una cadena $\omega \in \Sigma^*$ de tamaño n está o no en el lenguaje L' en tiempo polinómico $O(n^k)$ para algún $k \in \mathbb{N}$. Puesto que $L \preceq_p L'$, existe una función computable $f : \Sigma^* \to \Sigma^*$ y una máquina de Turing M_p tal que sus salidas son elementos de la forma $f(u)$ para algún $u \in \Sigma^*$ de tamaño m con tiempo de complejidad $O(m^r)$, siendo $r \in \mathbb{N}$.

Para saber si una cadena ω de tamaño n está en el lenguaje L, se calcula $f(\omega)$ en tiempo polinómico $O(m^r)$ usando la máquina M_p, luego usando la complejidad de la máquina M'_D se decide si $f(\omega) \in L'$ en tiempo $O((n^r)^k)$. Dado que la composición de polinomios es un polinomio se concluye que M_p a través de M'_D decide a $\omega \in L$ en tiempo polinomial y en consecuencia, $L \in$ **P**.

b) La demostración es similar al inciso a). Supongamos que $L' \in$ **NP** y que $f : \Sigma^* \to \Sigma^*$ es una función de cadena que reduce al lenguaje L al L' en tiempo polinomial. Si una cadena ω está en L, entonces podemos calcular $f(\omega)$ en tiempo polinomial y usar una máquina de Turing no determinista con una cota temporal polinómica para L', y comprobar si $f(\omega) \in L'$. Puesto que la composición de dos polinomios es también un polinomio, se concluye que $L \in$ **NP**. ∎

Definición 3.10. *Un lenguaje L es llamado* **NP***-difícil si para todo lenguaje $L' \in$ **NP**, L' puede ser reducido en tiempo polinómico al lenguaje L. Si $L \in$ **NP** y además es* **NP***-difícil, entonces L es llamado lenguaje* **NP***-Completo.*

Con las nociones anteriores estamos en condiciones de describir el problema de satisfacibilidad formulas booleanas como sigue:

Sea $V = \{v_1, v_2, \ldots, v_n\}$ un conjunto de variables. Una fórmula booleana es una expresión compuesta de elementos en V y operadores lógicos $\neg$, $\vee$ y $\wedge$. La expresión $\bigvee_{i=1}^{k}(v_i)$ es conocida como la cláusula de las variables v_i y la expresión $\bigwedge_{j=0}^{k}(\bigvee_{i=0}^{k}(v_{ij}))$ es denominada como fórmula normal conjunta, donde $i, j \in \{1, 2, 3, \ldots, k\}$.

Si φ es una fórmula sobre las variables $v_1, v_2, \ldots, v_n$ y $s \in \{0,1\}^n$ son los valores de verdad para cada v_i, entonces $\varphi(s)$ denota el valor de verdad de la fórmula. Por ejemplo la fórmula normal conjunta $\varphi = (v_1 \vee v_3) \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee v_3)$ es verdadera para los valores $s = (1,1,0) \in \{0,1\}^3$.

En la tabla 3.3, se observan los diferentes valores de verdad $s_1, s_2, \ldots, s_8$ para los cuales la fórmula $\varphi = (v_1 \vee v_3) \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee v_3)$ es verdadera o falsa. Específicamente se puede notar que los valores s_2, s_6, s_7 y s_8 hacen que el valor de verdad de la fórmula φ sea 1, en este caso decimos que φ es factible para los valores antes mencionados. En general, una fórmula φ es factible si existe un conjunto de valores $s \in \{0,1\}^n$ tal que $\varphi(s) = 1$.

En la fórmula general $\varphi = \bigwedge_{j=0}^{k}(\bigvee_{i=0}^{k}(v_{ij}))$, también observamos que hay a lo más n variables de las que se obtienen un total de 2^n combinaciones posibles de los valores booleanos para obtener el valor de verdad de la fórmula φ. Dado que el número de combinaciones posibles es exponencial, un procedimiento que determine los valores de verdad s_i para los cuales $\varphi(s_i) = 1$ en un tiempo razonable es impensable.

	v_1	v_2	v_3	$(v_1 \vee v_3)$	$(v_1 \vee \neg v_2)$	$(v_2 \vee v_3)$	$\varphi(s_i)$
s_1	0	0	0	0	1	0	0
s_2	0	0	1	1	1	1	1
s_3	0	1	0	0	0	1	0
s_4	0	1	1	1	0	1	0
s_5	1	0	0	1	1	0	0
s_6	1	0	1	1	1	1	1
s_7	1	1	0	1	1	1	1
s_8	1	1	1	1	1	1	1

Tabla 3.3: Valores de verdad de $\varphi(s_i)$.

Sin embargo, el modelo no determinista de máquinas de Turing proporciona una muy buena alternativa para analizar este tipo de problemas, para ello denotaremos al problema de fórmulas normales conjuntas por $SAT = \{< \varphi > \mid \varphi(s_i) = 1\}$ por sus primeras tres letras de su nombre en inglés, donde $< \varphi >$ es la codificación de una fórmula normal conjunta factible.

Teorema 3.4. *El problema SAT pertenece a la clase* **NP***-Completa.*

Para demostrar que *SAT* es **NP**-Completo, se debe demostrar primero que *SAT* es **NP**, para ello se requiere de la construcción de una máquina de Turing no determinista de dos cintas con complejidad temporal de tipo polinomial. La construcción de tal mecanismo procesa las codificaciones adecuadas de las fórmulas φ del problema *SAT*.

Si $\{v_1, v_2, \ldots, v_n\}$ es un conjunto ordenado de variables booleanas, cada variable v_i será codificada por medio de su índice en forma binaria, por ejemplo, a la variable v_5 le corresponde la cadena binaria 101, si la variable es positiva la codificación completa de v_5 es la cadena 101#1, donde #1 indica que la variable es positiva. Por otro lado la codificación de la variable $\neg v_3$ se expresa mediante la cadena 11#0, claramente #0 indica que la variable es negativa.

La codificación de una fórmula φ utiliza el alfabeto finito $\Sigma = \{0, 1, \bullet, \#, \vee, \wedge\}$, por ejemplo la fórmula $\varphi = (v_1 \vee v_3) \wedge (v_1 \vee \neg v_2) \wedge (v_2 \vee v_3)$ presentada anteriormente se expresa mediante la cadena $1\#1 \vee 11\#1 \wedge 1\#1 \vee 10\#0 \wedge 10\#1 \vee 11\#1$. Finalmente la codificación completa de la fórmula φ sobre la cinta 1 de la máquina de Turing no determinista tendrá la forma,

$$\triangleright \underbrace{1 \bullet 10 \bullet 11}_{\text{Variables}} \bullet\bullet \underbrace{1\#1 \vee 11\#1 \wedge 1\#1 \vee 10\#0 \wedge 10\#1 \vee 11\#1}_{\text{Fórmula}} \triangleright$$

La cadena $1 \bullet 10 \bullet 11$ contiene la secuencia ordenada de variables que conforman la fórmula φ codificadas en forma binaria y separadas una de la otra por el símbolo $\bullet$. Los símbolos $\bullet\bullet$ se implementan para indicar donde empieza la codificación de la fórmula φ. La cinta 2 de la máquina contiene una cadena con las asignaciones correspondiente a los valores $s \in \{0,1\}^n$, en particular si elegimos $s = \{1, 1, 0\}$, la cadena escrita sobre la cinta 2 tendrá la forma,

$$\triangleright\ 1\#1 \bullet 10\#1 \bullet 11\#0 \triangleright$$

El procedimiento correspondiente para analizar las fórmulas del problema *SAT* sobre una la máquina no determinista se describe a continuación, la idea básica de la prueba fue tomada de [41, pág. 463].

Demostración. Sea $V = \{v_1, v_2, \ldots, v_n\}$ un conjunto de variables booleanas, $C(v_i)$ la codificación binaria correspondiente a la variable $v_i \in V$ para algún $i \in \mathbb{N}$ y $s = \{B(v_1), B(v_2), \ldots, B(v_n)\}$ el conjunto de valores binarios, donde cada $B(v_i)$ toma un valor en $\{0, 1\}$.

La construcción de una máquina de Turing no determinista M que acepte el lenguaje *SAT* se describe a continuación:

1.- Inicialmente la máquina de Turing no determinista contiene la fórmula codificada sobre la primera cinta y la segunda cinta contiene espacios en blanco. Si la primera cinta no contiene la codificación correspondiente de una fórmula φ entonces la computación de la máquina se detiene y rechaza la cadena.

2.- La codificación de variable v_i es copiada sobre la segunda cinta de máquina asignándole el valor $B_M(v_i)$ correspondiente, es importante indicar que los valores $B_M(v_i)$ de la cinta dos son diferentes a los valores $B(v_j)$ en la cinta uno. Los valores $B_M(v_i)$ son conjeturados por la máquina no determinista mediante algún mecanismo definido. Cada codificación binaria $C(v_i)\#B_M(v_i)$ en la cinta dos es separada por el símbolo $\bullet$.

Cuando la cabeza de la primera cinta lee los símbolos $\bullet\bullet$, en ese instante, la cabeza de la segunda cinta es posicionada al principio de la cinta justo al iniciar la variable v_1 codificada y la cabeza de la primera cinta se posiciona al principio de la variable v_1 codificada después de los símbolos $\bullet\bullet$ tal como se aprecia a continuación,

$$\text{Cinta 2} \quad \triangleright \underbrace{C(v_1)}_{\text{Cabeza 2}} \#B_M(v_1) \bullet C(v_2) \#B_M(v_2) \bullet C(v_3) \#B_M(v_3) \bullet \cdots \bullet C(v_n) \#B_M(v_n) \triangleright$$

$$\text{Cinta 1} \quad \triangleright C(v_1) \bullet \cdots \bullet C(v_n) \bullet\bullet \underbrace{C(v_1)}_{\text{Cabeza 1}} \#B(v_1) \vee \cdots \vee C(v_l) \#B(v_l) \wedge \cdots \wedge C(v_1) \#B(v_1) \vee \cdots \vee C(v_k) \#B(v_k) \triangleright$$

3.- La máquina escanea las codificaciones de las variables $C(v_i)$ en la primera y segunda cinta, una vez realizado esto, la máquina compara los valores de verdad asignados a $B_M(v_i)$ y $B(v_i)$.

4.- Sí los valores de verdad de $B_M(v_i)$ y $B(v_i)$ no son idénticos, la variable actual no se satisface con el valor asignado y se continúa con el proceso, escaneando las siguientes variables. En caso de que el símbolo a continuación sea $\triangleright$ o $\wedge$, significa que todas las variables de la cláusula actual han sido examinadas y no se ha encontrado un emparejamiento en los valores de verdad y en consecuencia la máquina se detiene y rechaza.

5.- Sí los valores de verdad de $B_M(v_i)$ y $B(v_i)$ son idénticos y se continúa con el proceso de lectura y comparación. La cabeza de la cinta uno se mueve hasta el próximo símbolo $\wedge$ o $\triangleright$. Si se encuentra al espacio en blanco la computación se detiene y la máquina acepta, en caso contrario se continúa analizando la próxima cláusula a partir del paso 3.

Este proceso requiere comparar las variables en la cinta uno con cada variable de la cinta dos, comparar si sus valores son idénticos, se realiza en tiempo polinomial $O(kn^2)$, siendo n el número de variables y k el número de variables en la fórmula de entrada, concluyendo así que $SAT \in \mathbf{NP}$.

Para demostrar que SAT es **NP**-Completo, solo falta ver que SAT es **NP**-difícil, para ello debemos establecer que todo lenguaje **NP** se puede reducir al problema SAT. La prueba es extensa y no la describiremos aquí, ésta se puede encontrar detalladamente en [51, pág. 278] y [15]. ∎

Para ver que un lenguaje es **NP**-Completo solo hay que probar que existe una reducción en tiempo polinomial al lenguaje SAT o a cualquier lenguaje **NP**-Completo conocido. Esta característica importante de reducción entre problemas de la clase **NP**-Completa plantea la pregunta ¿Existe una reducción de tiempo polinomial de un lenguaje **NP**-Completo a un lenguaje de la clase **P**? La controversial pregunta surge a raíz de la reducción universal existente entre problemas de la clase **NP**-Completa para los cuales no existe un algoritmo (máquina de Turing) eficiente que los resuelva.

Entre los diversos problemas de naturaleza **NP**-Completa tenemos: el problema de la mochila, el problema de ciclos de una gráfica de Hamilton, el problema general de particiones, el problema 3-*SAT* y problema del agente viajero (véase [42, pág. 317]). Algunos problemas pertenecen a áreas multidisciplinarias, tal es el caso del problema de encontrar equilibrios de Nash en economía ([48], [56]), el problema del plegado de proteínas en biología [32], entre muchos otros. En caso de existir la reducción que plantea la pregunta, tendríamos que los problemas de la clase **NP** tienen un algoritmo eficiente que los decide, en este caso $\mathbf{P} = \mathbf{NP}$.

Teorema 3.5. *Si L es un lenguaje* **NP***-Completo y $L \in \mathbf{P}$, entonces* $\mathbf{P} = \mathbf{NP}$.

Demostración. Sea $L^{'}$ un lenguaje en **NP**, puesto que L es **NP**-Completo, existe una reducción polinomial tal que $L^{'} \preceq_p L$. Dado que $L \in P$, aplicando el inciso a) del teorema 3.3 obtenemos que $L^{'} \in P$. ∎

Las implicaciones de resultar cierto el teorema anterior conllevarían a resolver los problemas de la clase **NP** de forma eficiente, sin embargo de resultar lo opuesto, es decir, que $\mathbf{P} \neq \mathbf{NP}$, la incesante búsqueda de diseños algorítmicos eficientes sería innecesaria y los esfuerzos seguirían la dirección de mejorar las técnicas para atacar los problemas **NP** invirtiendo más en la capacidad de cómputo con el uso de procesadores. Hasta ahora no existe una prueba aceptada en consenso que demuestre si $\mathbf{P} = \mathbf{NP}$ o $\mathbf{P} \neq \mathbf{NP}$, los diferentes trabajos al respecto no dan certeza de qué dirección sea correcta, al respecto se puede consultar [65], en donde se puede acceder a una gran cantidad de artículos sobre los diferentes enfoques para establecer si $\mathbf{P} = \mathbf{NP}$ o $\mathbf{P} \neq \mathbf{NP}$.

Este problema se conoce como la conjetura **P** contra **NP** y forma parte de la lista de problemas del milenio propuesta por el instituto Clay de Matemáticas [45]. Demostrarla requiere de un nivel muy avanzado de estudios y posiblemente de una nueva técnica o teoría. El objetivo de presentar la conjetura es con la intención de establecer su relación con los algoritmos de calendarización. En el siguiente capítulo presentaremos una compilación de resultados de la teoría de la calendaricación, especialmente de modelos de máquinas secuenciales que están altamente ligados a esta famosa conjetura.

Capítulo 4

Complejidad de la Calendarización

Reducir un problema en otro, significa ver al primer problema desde la perspectiva del segundo con la intención de probar con soluciones conocidas.

Anónimo.

El problema de la calendarización es esencialmente uno de complejidad, lo cual puede ser atacado por una gran variedad de técnicas. Por el momento comenzaremos recordando algunas definiciones previamente establecidas en el capítulo anterior:

Definición 4.1 (Reducción de lenguajes y problemas)**.** *Dado un alfabeto Σ, se dirá que el lenguaje L_1 se puede* ***reducir*** *al lenguaje L_2 (el cual se denota $L_1 \preceq L_2$); si existe una máquina de Turing determinista, que convierte a todo elemento $x_1 \in L_1$ en un elemento $x_2 \in L_2$ y además se satisfacen las siguientes condiciones:*

- $x_1 \in L_1 \leftrightarrow x_2 \in L_2$
- *La conversión se realiza en tiempo polinomial (dependiendo de $n = |x_1|$).*

Recordemos que si $L_1 \preceq L_2$, entonces una máquina de Turing que decide si $x_2 \in L_2$ puede también utilizarse para decidir si $x_1 \in L_1$, por otro lado también significa que decidir si $x_2 \in L_2$ es **por lo menos tan difícil** como decidir si $x_1 \in L_1$.

Notese también que, como se vió en el capítulo anterior, se puede utilizar la misma definición al hablar de la complejidad de la solución de problemas de cálculo en general.

Ejemplo 4.1. *Un caso sencillo lo constituye el problema de decidir si un número es divisible por 2, el cual se puede reducir al problema de decidir si dicho número se puede dividir por 4, para lo cual basta con multiplicar el número por 2, que claramente requiere un tiempo lineal y por lo tanto polinomial.* □

Ejemplo 4.2. *Con frecuencia se considerarán los casos especiales de un problema aún más general, por ejemplo* $(\alpha|\beta|\Sigma \mathbf{C_j})$ *es un caso especial de* $(\alpha|\beta|\Sigma \mathbf{w_j C_j})$*, de manera que todo caso especial se puede claramente reducir al problema general.* □

Como veremos más adelante, una gran parte de los problemas de calendarización de una sola máquina determinista, son de la clase **NP**-difíciles, la cual recordaremos a continuación.

Definición 4.2. *Se dice que L es* **NP***-difícil si todo lenguaje en* **NP** *se puede reducir determinísticamente en tiempo polinomial a L.*

La definición anterior significa que L es por lo menos tan díficil como cualquier problema en **NP**. Los lenguajes de la clase **NP** se consideran generalmente **intractables** en el sentido de que es poco probable que se encuentre un procedimiento de decisión eficiente (determinístico en tiempo polinomial).

Ejemplo 4.3. *Más adelante se verá que el problema* $(1|r_j|L_{max})$ *es* **NP***-completo, el cual es un caso particular del problema* $(1|r_j, prec|L_{max})$ *por lo que se deduce que éste último también es* **NP***-completo.* □

Teorema 4.1. *Dados dos problemas de calendarización* $(\alpha|\beta|C_{max})$ *y* $(\alpha|\beta|L_{max})$*, donde* α *y* β *pueden ser arbitrarias, pero iguales para ambos problemas, entonces se cumple:*

$$(\alpha|\beta|C_{max}) \preceq (\alpha|\beta|L_{max})$$

Demostración. Trivialmente, a cada caso de $(\alpha|\beta|C_{max})$ se le asocia el problema correspondiente de $(\alpha|\beta|L_{max})$, en donde para todo trabajo j se cumple $d_j = 0, j = 1, \ldots, n$, entonces se tiene:

$$\begin{array}{rcll} L_{max} & = & \max\{L_1, \ldots, L_n\} & = \max\{C_1 - d_1, \ldots, C_n - d_n\} \\ & = & \max\{C_1 - 0, \ldots, C_n - 0\} & = \max\{C_1, \ldots, C_n\} \\ & = & C_{max} & \end{array}$$

Con este simple teorema, podemos concluir que el problema de calendarización $(\alpha|\beta|L_{max})$ siempre es por lo menos tan díficil como $(\alpha|\beta|C_{max})$. Si además nos es conocido que para determinados valores de α y β, el problema $(\alpha|\beta|C_{max})$ es **NP**-difícil, entonces el problema $(\alpha|\beta|L_{max})$ también lo es. Si en cambio, el problema $(\alpha|\beta|L_{max})$ esta en **P**, entonces el problema $(\alpha|\beta|C_{max})$ también está en **P**.

Más generalmente, veremos el siguiente teorema:

Teorema 4.2 (Reducción de Problemas de Calendarización). *Para valores arbitrarios pero fijos de* α *y* β *se cumple:*

(1) $(\alpha|\beta|L_{max}) \preceq (\alpha|\beta|\sum T_j)$

(2) $(\alpha|\beta|L_{max}) \preceq (\alpha|\beta|\sum U_j)$

(3) $(\alpha|\beta|\sum C_j) \preceq (\alpha|\beta|\sum T_j)$

(4) $(\alpha|\beta|\sum w_j C_j) \preceq (\alpha|\beta|\sum w_j T_j)$

Demostración. 1 y 2. El método de reducción se aplicará de la manera siguiente: Dado un problema $(\alpha|\beta|L_{max})$ y un valor $n \in \mathbb{N}$ se reducirá la cuestión de si hay una calendarización tal que $L_{max} \leq n$, a la cuestión de si hay un valor $m \in \mathbb{N}$ con $m = \phi(n)$ para la cual $\sum T_j \leq m$, pues como se observó al final del capítulo 1, todo problema de optimización se puede reducir a uno de decisión.

Si $n = 0$, entonces $L_{max} \leq 0$ es equivalente a $\sum T_j = \sum U_j = 0$, por lo que podemos suponer que $n > 0$. En tal caso transformamos primeramente el problema inicial en uno cuya única diferencia es que las fechas de entrega se recorren n unidades hacia atrás. Entonces el problema original tiene un máximo retraso de n unidades de manera que también se cumple $\sum T_j = \sum U_j = 0$ como en el caso anterior.

3. y 4. La demostración es análoga al teorema anterior haciendo $d_j = 0, j = 1, \ldots, n$ para todos los trabajos y la equivalencia es clara. ∎

Representando por una flecha a la relación de reducción $\preceq$, los resultados del teorema anterior se pueden representar de la manera siguiente:

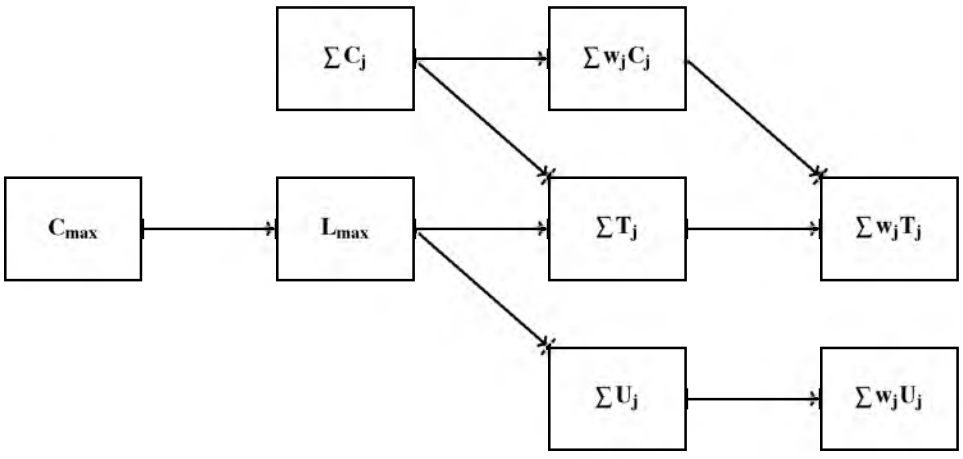


Figura 4.1: Jerarquía de complejidad.

4.1. Modelos de una sola máquina

Para efectos de la clasificación siguiente, recordemos que la complejidad de un *sort* (véase, [60]) es del orden $O(n \cdot \log n)$. Por los últimos dos teoremas de la sección anterior, se concluye que también $1||L_{max}$ y $1||C_{max}$ también estan en la clase **P** y son por lo tanto de fácil solución, más aún, se puede concluir que el caso particular del campo β con $p_j = p, d_j = d$ también son de la misma clase.

Para los tres problemas restantes, suponiendo el campo β vacío, se tiene el siguiente teorema:

Teorema 4.3. *Los problemas* $1||\sum T_j, 1||\sum w_j T_j$ *y* $1||\sum w_j U_j$ *son todos de la clase* **NP***-difícil.*

Demostración. La demostración no es simple, por lo que se hace referencia a la literatura (ver. Du y Leung 1990). En el se encuentra la demostración de que $1||\sum T_j$ es **NP**-díficil, de lo cual se sigue directamente el caso $1||\sum w_j T_j$. Para una demostración de que $1||\sum w_j U_j$ es **NP**-difícil véase Lawler y Moore [14], así como Karp [17].

El caso $1||\sum w_j U_j$, con $1|d_j = d|\sum w_j U_j$, se puede reducir al problema de la mochila; en el cual, el tiempo de entrega corresponde a la capacidad de la mochila y el tiempo de proceso de un pedido corresponde al espacio que ocupa una pieza en la mochila. A su vez, los pesos asignados reflejan la ganancia de cada parte. ∎

Generalizaciones de los problemas descritos en el teorema anterior son en su totalidad **NP**-difícil y se obtienen agregando condiciones complementarias que pueden tenerse a mano, las cuales se encuentran dadas en el campo β y normalmente son $r_j, prec$ y s_{jk} así como cualquier combinación de estas.

4.1.1. Minimización de C_{max}

Teorema 4.4. *El problema* $1|r_j|C_{max}$ *se puede resolver en* $O(n \cdot log\ n)$ *es decir, es de clase* **P**.

Demostración. La afirmación es evidente y se hace siguiendo la misma técnica delineada en el capítulo 1. Ordenando los trabajos en forma creciente de r_j, veremos entonces que el calendario $S(j) = j$ es óptimo.

Sea S_1 un calendario para el cual se cumple $\exists k \in \{1, \ldots, n\}$ tal que $S_1(k+1) = S_1(k) + 1$, es decir la orden k se ejecuta inmediatamente antes que la orden $k+1$ a pesar de que $r_j < r_{k+1}$, mientras que en el calendario S, la orden k se ejecuta antes que la $k+1$.

Sea S_T el tiempo de inicio del trabajo k en el calendario S, para él se tiene $S_T \geq r_k$ y además:

$$\begin{aligned} C_k &= S_T + p_k \geq r_k + p_k \\ C_{k+1} &= C_k + p_{k+1} \geq r_k + p_k + p_{k+1} \end{aligned}$$

en el calendario S_1, la orden $k+1$ se ejecuta antes de la orden k cuyo tiempo de inicio S_1^T es mayor o igual que r_{k+1}, de manera que los tiempos de terminación C_k^1 y C_{k+1}^1 son:

$$\begin{aligned} C_{k+1}^1 &= S_1^T + p_{k+1} \geq r_{k+1} + p_{k+1} \\ C_k^1 &= C_{k+1}^1 + p_k \geq r_{k+1} + p_{k+1} + p_k \end{aligned}$$

de donde se sigue que $C_k^1 \geq C_{k+1}$ y por lo tanto C_{max} es mayor en S_1 que en S. ∎

Teorema 4.5 ($1|\mathbf{prec}|\mathbf{C_{max}}$). *El problema $1|prec|C_{max}$ se puede resolver en n^2 pasos, por lo que está en la clase* **P** *y además para cualquier relación de dependencia entre los trabajos, la función objetivo siempre se optimiza como*

$$C_{max} = \sum_{j \in \{1,\ldots,n\}} p_j.$$

También en este caso existe un procedimiento que proporciona de manera clara el calendario óptimo.

Algoritmo 4 ($1|\mathbf{prec}|\mathbf{C_{max}}$)

1. Inicialización : Sea $J = \{1,\ldots,n\}$ el conjunto de los pedidos que faltan por calendarizarse.

2. Establecimiento del próximo pedido : Elijase un pedido $j \in J$, para el cual no hay otro pedido en J que deba ejecutarse antes de j.

Si no existe tal pedido, entonces debe existir un ciclo en la gráfica de precedencia y el problema no se puede resolver.

3. Recursión : Planificar j como el siguiente pedido y eliminarlo de J. Si J esta vacío terminar, en caso contrario regresar al paso 2.

El siguiente es un caso del problema $(1|\beta|C_{max})$ que es del tipo **NP**.

Teorema 4.6. $(1|s_{jk}|C_{max})$ *es* **NP**.

Demostración. Recordar que el campo s_{jk} indica que hay un tiempo de preparación de la máquina, el cual depende del orden de los procesos.

Para demostrar que el problema es **NP**-difícil, se reducirá a uno de esa clase. Para ello tomamos el problema del agente viajero que se sabe es **NP**-difícil, se sabe además que el problema es también **NP**-difícil cuando se impone la restricción que el agente no debe regresar al lugar de partida al final del viaje. Esta variante se puede reducir al problema $(1|\beta|C_{max})$ de la siguiente manera: para cada una de las ciudades que se deben visitar, se genera un pedido de la calendarización. Las distancias entre las ciudades corresponden a los tiempos de preparación de la máquina. Por lo que un calendario óptimo de $(1|\beta|C_{max})$ corresponde al camino más corto que resuelve el problema del agente. ∎

4.1.2. Minimización de $\Sigma w_j C_j$

Aunque $(1||\sum C_j)$ y su generalización $(1||\sum w_j C_j)$ se pueden resolver fácilmente mediante un ordenamiento, no sucede la mismo para generalizaciones del campo β. Sin entrar en detalles de la demostración, en seguida se dan los principales resultados:

Teorema 4.7. *Los problemas* $(1|r_j|\sum C_j)$ *y* $(1|prec|\sum C_j)$, *son ambos* **NP**-*difíciles.*

Demostración. Para $(1|r_j|\sum C_j)$, véase Lenstra et al. [20] y para la demostración de $(1|prec|\sum C_j)$ en Lenstra and Rinnoy Kann [25]. ∎

Del teorema anterior se vé fácilmente que también $(1|r_j|\sum w_j C_j)$ y $(1|r_j|\sum w_j C_j)$ son **NP**-difíciles.

Tomando algunas ideas del teorema (4.5), se puede demostrar lo siguiente:

Teorema 4.8. *El problema* $(1|s_{jk}|\sum C_j)$ *es* **NP**-*difícil.*

Demostración. Demostraremos que el caso especial $(1|s_{jk}, p_j = 1|\sum C_j)$ es **NP**-difícil. Para lo cual utilizaremos también un caso particular (**NP**-difícil) del problema del agente viajero, en el cual, la máxima distancia entre dos ciudades es menor que cierto entero $b \in \mathbb{N}$. Partiendo de la variante del problema en la cual el agente no debe necesariamente regresar al lugar de origen. Sea m el número de ciudades a visitar en un problema del agente viajero, al cual vamos a reducir el problema $(1|s_{jk}|\sum C_j)$.

En el problema de calendarización habrá en total $n = bm^3$ pedidos, los primeros m a los cuales llamaremos pedidos "delanteros", corresponden a las ciudades, de manera que (análogamente a 4.5) los tiempos de preparación entre dos pedidos corresponden justamente a la distancia $dist(j,k)$. En general, se cumple lo siguiente:

$$\mathbf{s}_{jk} := \begin{cases} dist(j,k) & \text{si} \quad 1 \leq j,k \leq m \\ 0 & \text{si} \quad 1 \leq j \leq m < k \leq n \\ 2(bm)^6 & \text{si} \quad 1 \leq k \leq m < j \leq n \\ 0 & \text{si} \quad m \leq j,k \leq n \\ 0 & \text{si} \quad j = 0 \text{ y } 1 \leq k \leq n \end{cases}$$

El tiempo de preparación de uno de los pedidos delanteros a uno de los pedidos traseros siempre es 0 (segunda línea), mientras que en orden inverso, se aplica un tiempo de preparación muy alto $2(bm)^6$, los demás tiempos de preparación son iguales a 0. Finalmente, todos los tiempos de procesamiento son iguales a 1.

En seguida notamos la siguiente propiedad de un calendario óptimo: un pedido trasero nunca se puede colocar directamente antes de uno de los delanteros. Si fuera así, la función objetivo sería mucho mayor que $2(bm)^6$. Pero claramente, el calendario óptimo tiene una función objetivo menor que ese valor. Consideremos ahora el tiempo total de procesamiento C_{max} en el caso en que todos los pedidos delanteros se ejecutan primeramente. Entonces, entre los primeros m pedidos hay tiempos de preparación cuya suma es menor o igual que $b(m-1)$. Considerando los tiempos de preparación, C_{max} es menor o igual que $b(m-1+m^3)$ y puesto que $C_j \leq C_{max} \leq b(m-1+m^3) < 2(bm)^3$ y hay m pedidos, la función objetivo $\sum C_j$ es seguramente menor que $2(bm)^6$.

Ahora ya sabemos que todos los pedidos delanteros se deben ejecutar al principio. Por otro lado se debe notar que $\sum C_j$ solo es mínimo cuando C_{max} lo es, lo cual se justifica de la siguiente manera: el último de los primeros pedidos (el cual determina C_{max}) tiene un tiempo de terminación menor ó igual que bm, la suma de los tiempos de terminación de los primeros m pedidos es por lo tanto menor que bm^2. De manera que si se puede reducir C_{max} por una unidad, se reducen los tiempos de terminación de los pedidos traseros también por una unidad, lo cual da en total bm^2 unidades.

Se debe entonces minimizar el tiempo total, lo cual sucede cuando el problema correspondiente del agente viajero se minimiza. ∎

De lo anterior, se concluye que el caso ponderado $(1|s_{jk}|\sum w_j C_j)$, que es más general que el tratado en el teorema, también es **NP**-difícil.

Ahora consideraremos otro problema que se puede resolver de manera simple, a saber $(1|s_{jk}, prmtn|\sum C_j)$. En un problema planteado de esta manera, se permite interrumpir el procesamiento de un pedido, en particular cuando se presenta otro con menor tiempo de procesamiento. En este caso, los pedidos se ordenan en forma creciente por el menor tiempo de procesamiento restante (SRPT-shortest remaining processing time). El algoritmo correspondiente que se muestra a continuación nos da el calendario de procesamiento óptimo.

Algoritmo 5 $(\mathbf{1|r_j, prmtn|\sum C_j})$

1. Inicialización : Sea $J = \{1,\ldots,n\}$ el conjunto de los pedidos que faltan por calendarizarse. Más aún, sean $p_j^r = p_j, \forall j \in J$ el tiempo restante de procesamiento del pedido j. El momento inicial es $t = 0$.

2. Establecimiento del próximo pedido : Elijase un pedido

$$j^* \in \min_{j\in\{1,\ldots,n\}, r_j \leq t} p_j^r$$

es decir, el pedido, con el menor tiempo de procesamiento entre los disponibles. Sea

$$r^{(min)} := \min_{j\in\{1,\ldots,n\}, r_j > t}$$

el momento más cercano en que un pedido está disponible.

Si j^* existe, hágase $t = r^{(min)}$ e ir al paso 2. Si $r^{(min)} < t + p^r_{j^*}$ (El pedido j^* no se puede completar antes antes de que se termine algún otro), ir al paso 3, en caso contrario ir al paso 4.

3. Interrupción : Ejecutar el pedido j^* hasta el momento $r^{(min)}$ e interrumpir el procesamiento. Hacer $p^r_{j^*} = p^r_{j^*} - r^{(min)} + t$ y $t = r^{(min)}$. Ir al paso 2.

4. Terminación de un pedido : Ejecutar el pedido j^* hasta su terminación. Hacer $t = t + p^r_{j^*}$ y eliminar j^* de J. Si J no es vacío ir al paso 2.

Ejemplo 4.4. *Consideremos el siguiente caso del problema* $(\mathbf{1}|\mathbf{r_j}, \mathbf{prmtn}|\sum \mathbf{C_j})$

j	1	2	3	4
p_j	6	8	3	4
r_j	0	0	2	14

Al aplicar el algoritmo definido anteriormente, se tiene que al principio solo los pedidos 1 y 2 están disponibles. Se calendariza primero el pedido 1, pues es el que tiene el menor tiempo de procesamiento. El pedido 3 esta disponible en el momento 2, de manera que nos encontramos en el punto 3 del algoritmo por lo que el proceso 1 solo se ejecuta hasta el momento 2. Lo cual significa que nos encontramos en el momento 2 y al pedido 1 le faltan 4 unidades para su terminación.

Entre los tres pedidos restantes, el no. 3 tiene el menor tiempo de procesamiento, por lo que se puede ejecutar completamente (paso 4 del algoritmo). Enseguida se termina el pedido 1, de forma que nos encontramos en el momento 9 y solo faltan por terminarse los pedidos 2 y 4. Aquí se debe notar que el pedido 2 es el único disponible hasta el momento 14 y en ese tiempo todavía le faltan 3 unidades de procesamiento por lo que se continúa procesando hasta el final y en seguida se procesa el pedido 4.

El calendario óptimo se muestra en el siguiente diagrama de Gantt.

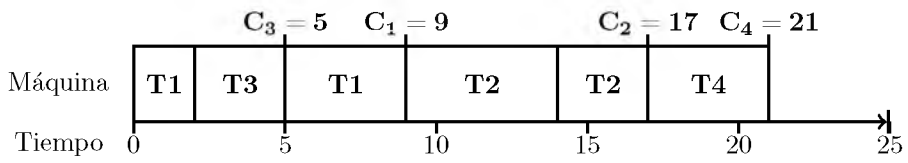


Figura 4.2: Calendario óptimo del ejemplo 4.7.

Se puede ver que la generalización del problema con la suma ponderada $(1|r_j, pmtn|\sum w_j C_j)$ *es* **NP***-difícil, véase Labetoulle el al. [27].* □

4.1.3. Funciones Objetivo Dependientes de L, T y U

Considerémos primero la función objetivo L_{max}. Puesto que C_{max} es un caso especial de ellos, significa que los problemas que son difíciles con la función objetivo C_{max} también lo son con la función L_{max}. En particular, el problema $(1|s_{jk}|L_{max})$ es **NP**-difícil porque $(1|s_{jk}|C_{max})$ lo es.

En cambio, el problema $(1|prec|L_{max})$ es fácil de resolver, lo cual se logra con una extensión del Algoritmo 4.

Algoritmo 6 $(1|\mathbf{prec}|\mathbf{L_{max}})$

1. Inicialización : Sea $J = \{1,\ldots,n\}$ el conjunto de los pedidos que faltan por calendarizarse.

2. Establecimiento del próximo pedido : Elijase un pedido $j \in J$, para el cual no hay otro pedido en J que deba ejecutarse antes de j y cuyo tiempo de entrega sea el menor.

Si no existe tal pedido, entonces debe existir un ciclo en la gráfica de precedencia y el problema no se puede resolver.

3. Recursión : Planificar j como el siguiente pedido y eliminarlo de J. Si J esta vacío terminar, en caso contrario regresar al paso 2.

Si en cambio se debe considerar los tiempos de llegada r_j el problema se vuelve **NP**-difícil, su demostración se puede encontrar en (Lenstra el al. 1977), la cual incluíremos aquí, debido por una parte a que es un problema prototipo de demostración de complejidad y por otra parte a que utiliza la reducción a uno de los clásicos problemas **NP**-difícil: el problema de partición 3.

Teorema 4.9 $(1|\mathbf{r_j}|\mathbf{L_{max}})$. *El problema* $1|r_j|L_{max}$ *es* **NP***-difícil.*

Demostración. Para demostrar que se trata de un problema **NP**-difícil, se reducirá a una problema de partición 3, el cual se define como sigue:

CASO: Dado un problema A con justamente $3m, m \in \mathbb{N}$, tal que para todo elemento $a \in A$, se define su "tamaño" $p_a \in \mathbb{N}$, de forma que $B := \frac{\sum_{a\in A} p_a}{m}$ sea un entero con $\frac{B}{4} < p_a < \frac{B}{2}$.

CUESTION: ¿Es posible encontrar una partición de A en m subconjuntos $S_1,\ldots,S_m$ de forma que $\sum_{a\in S_i} p_a = B$ para todo $i \in \{1,\ldots,m\}$?, lo cual significa que cada subconjunto S_i consta de justamente 3 elementos.

Así que partiendo de un problema de partición 3, construiremos un problema $1|r_j|L_{max}$, para el cual existe un calendario con $L_{max} = 0$ sí y solo si el problema de partición 3 es decidible.

Para el problema $1|r_j|L_{max}$ se tendrán $4m-1$ pedidos, de los cuales a los últimos $m-1$ pedidos, es decir, los pedidos $3m+1, 3m+2,\ldots,4m-1$ los llamaremos los "bloques "para los cuales su tiempo de procesamiento es B, es decir:

$$p_a = B, \quad \forall a \in \{3m+1,\ldots,4m-1\}$$

Los tiempos de llegada son:

$$r_{3m+1} = B, r_{3m+1} = 3B, r_{3m+1} = 5B \quad \text{etc.}$$

Los tiempos de entrega son:

$$d_{3m+1} = 2B, d_{3m+1} = 4B, d_{3m+1} = 6B \quad \text{etc.}$$

La relación entre los tiempos de llegada y entrega es tal que los pedidos deben procesarse inmediatamente de forma que se obtenga $L_{max} = 0$. El orden de los $m-1$ bloques se representa en la siguiente figura:

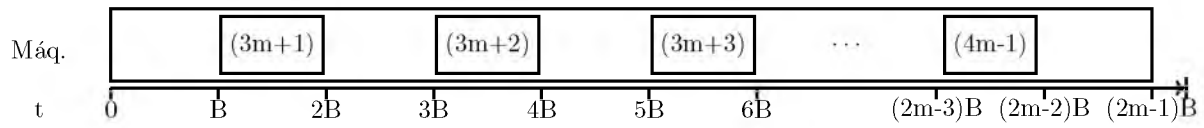


Figura 4.3: Calendario S(j)=j.

El resto de los $3m$ pedidos tienen un tiempo de proceso dados justamente por el caso del problema de partición 3, o sea: $p_j, j \in \{1,\ldots,3m\}$, todos estos pedidos tienen tiempo de llegada 0, es decir $r_j = 0, j \in \{1,\ldots,3m\}$, y sus tiempos de llegada son $d_j = (2m-1)B, j \in \{1,\ldots,3m\}$. Si el caso de partición 3 es un caso afirmativo (SI), entonces se deben planear los bloques justamente en el tiempo como lo ilustra la figura y los otros pedidos se pueden colocar en las aberturas entre los bloques, de manera que el problema $1|r_j|L_{max}$ posee un calendario con $L_{max} = 0$. Por otra parte, si existe un calendario de $1|r_j|L_{max}$ con $L_{max} = 0$, entonces se utilizan los espacios entre los bloques completamente y en cada espacio se procesan justamente tres pedidos. De esta manera se obtiene una partición del problema de particion-3 que muestra que se trata de un caso positivo.

Si este problema se transforma de manera que se permitan interrupciones, entonces se trata de un problema con solución en tiempo polinomial. Por el momento no veremos explícitamente el algoritmo para $1|r_j, pmtn|L_{max}$, pues es esencialmente el algoritmos del esquema 3. La única diferencia es que en el paso 2, no se elige el pedido con el tiempo restante de procesamiento más corto, sino el pedido con el más próximo tiempo de entrega, es decir: $j^* \in \min_{j \in J, r_j \leq t} d_j$.

Consideremos ahora como función objetivo el número de pedidos retrasados $\sum U_j$. Puesto que este problema es una generalización de L_{max}, claramente los problemas $1|r_j|\sum U_j$ y $1|s_{jk}|\sum U_j$ son ambos NP-difíciles. Solo queda una generalización:

Teorema 4.10. $(\mathbf{1|prec|\sum U_j})$. *El problema* $(\mathbf{1|prec|\sum U_j})$ *es* **NP***-difícil.*

Demostración. El caso especial, en que la relación de precedencia se presenta como cadenas (es decir solo hay una tarea inicial y una final) y además los tiempos de procesamiento son iguales, también es **NP**-difícil. La demostración de este teorema denotado como $(\mathbf{1|chains, p_j = p|\sum U_j})$ se encuentra en Lenstra y Rinnooy Kan [25]. ∎

A diferencia de los problemas de clase **P** que se presentaron en el capítulo 1 y estaban caracterizados por su simplicidad, en este párrafo, no presentaremos ningún algoritmo para el problema $(\mathbf{1}|\mathbf{r_j}, \mathbf{pmtn}|\sum \mathbf{U_j})$, en el trabajo de Lawler [28] se presenta un algoritmo basado en programación dinámica, el cual está fuera de los propósitos de este trabajo.

Después de haber investigado el efecto de generalizaciones en problemas simples de una sola máquina, dirigiremos nuestra atención ahora a las tres funciones objetivo "difíciles": $\sum w_j U_j$,$\sum T_j$ y $\sum w_j T_j$ y trataremos de ver si se pueden resolver de manera fácil en algunos casos especiales.

Para la función objetivo $\sum w_j U_j$, el caso especial de tiempo de entrega único $(\mathbf{1}|\mathbf{d_j} = \mathbf{d}|\sum \mathbf{w_j U_j})$, como ya se ha visto, es equivalente al problema de la mochila y por lo tanto difícil. En cambio, en el caso de que los tiempos de procesamiento sean idénticos, existe un algoritmo de tiempo polinomial que siempre proporciona un calendario óptimo. Para este propósito se necesita no solamente que los tiempos de procesamiento sean idénticos, también deben *coincidir* en peso, en el sentido que se específica abajo:

Definición 4.3. ***Tiempos de procesamiento y pesos coincidentes.*** *En un problema de calendarización, se dice que los tiempos de procesamiento y pesos coinciden (ó también que son* agradables*), si para cada par de trabajos j y k con $p_j < p_k$ se cumple entonces que $w_j \geq w_k$. En otras palabras, si el tiempo de procesamiento de un pedido es menor que él de otro, entonces su peso no debe ser menor que él del otro.*

El procedimiento es fuertemente similar al de Moore (Algoritmo 1), por cierto que en el paso 3 se debe precisar exactamente, cual de los pedidos se debe recorrer hasta el final. En este caso es otra vez el trabajo más largo. Si éste es el más largo, entonces tiene el peso más pequeño, pues se supone que los tiempos de procesamiento y los pesos son *agradables*. En caso de que haya varios pedidos de la misma duración (por ejemplo, cuando $p_j = p$ para todos los trabajos), entonces se elige entre estos pedidos el que tenga el menor peso.

Algoritmo 7 $(\mathbf{1}|\mathbf{p_j} = \mathbf{p}|\sum \mathbf{w_j U_j}, \mathbf{1}|\mathbf{agradable}|\sum \mathbf{w_j U_j})$

1. Inicialización: Ordenar los trabajos en forma no decreciente de acuerdo a su tiempo de entrega, de manera que se tiene: $d_1 \leq d_1 \cdots \leq d_n$, entonces el calendario inicial es $S(j) = j, \forall j \in \{1,\ldots,n\}$ y se hace $U := 0$.

2. Criterio de fin: Si $C_j \leq d_j, \forall, j \in \{1,\ldots,n\}$ que satisfacen $S(j) \leq n - U$, terminar.

3. Determinación del trabajo que se va a recorrer:

Sea $k := argmin\{S(j)|C_j > d_j, S(j) \leq n - U, j \in \{1,\ldots,n\}\}$ el trabajo que primero se programa dentro de todos los que estan retrasados. Denótese por $J^{(k)} := \{j \in \{1,\ldots,n\}|1 \leq S(j) \leq S(k)\}$ al conjunto de los primeros $S(j)$ trabajos. El conjunto $L := \{j \in J^{(k)}|p_j = \text{máx}\{p_{j'}|j' \in J^{(k)}\}\}$ denota todos los pedidos que tienen el mayor tiempo de procesamiento entre los primeros $S(k)$ pedidos. Entre éstos, sea $l \in argmax\{w_j|j \in L\}$ un trabajo con el menor peso.

4. Recorrimiento del trabajo: Definase el siguiente calendario:

$$S'(j) := \begin{cases} n & \text{Si} \quad j = l \\ S(j) - 1 & \text{Si} \quad S(l) < S(j) \leq n \\ S(j) & \text{Otro caso} \end{cases}$$

Hacer $S := S', U := U + 1$ e ir al paso 2.

Para la función objetivo $\sum T_j$, trataremos los casos especiales del mismo tiempo de procesamiento y el mismo tiempo de entrega.

El caso $(1|p_j = p|\sum T_j)$ es de fácil solución, lo cual se deduce del hecho de que $(1|p_j = p|\sum w_j T_j)$ es de fácil solución, como se mostrará utilizando resultados del siguiente problema.

El problema de la asignación: Dados dos conjuntos A y B de la misma cardinalidad, digamos $n = |A| = |B|$, de manera que al elemento $j \in A$ se le asigna el elemento $k \in B$ y un costo C_{jk}, se trata de encontrar la asignación que minimiza la cantidad $\sum_{j \in A} C_{jk}$. Se sabe que este problema esta en la clase **P**.

Considerando ahora el problema $(1|p_j = 1|\sum w_j T_j)$, éste se puede reducir a un problema de asignación en el que a cada pedido se le asocia su posición en el calendario, es decir, el conjunto A consiste de los pedidos y el conjunto B de las posiciones de cada pedido en el calendario. Los "costos" C_{jk} de una tal asignación del pedido j a la posición k se pueden considerar como la contribución de ese pedido a la función objetivo, es decir $C_{jk} := w_j T_j$, mediante esta transformación (que se puede efectúar en tiempo polinomial), se demuestra que ambos problemas son equivalentes.

El argumento se puede extender sin dificultad al problema $(1|p_j = p|\sum w_j T_j)$ por lo que esta también en **P**.

Solo queda por resolver el problema $(1|d_j = 1|\sum T_j)$:

Teorema 4.11 $(1|\mathbf{d_j = d}|\sum \mathbf{T_j})$**.** *El problema* $(1|d_j = d|\sum T_j)$ *se puede resolver en* $O(n \log n)$ *pasos, el calendario óptimo se obtiene ordenando los pedidos en orden creciente del tiempo de procesamiento (SPT: Shortest Processing Time).*

Demostración. Se debe notar primeramente, que debido a que el tiempo de entrega d es el mismo para todos, el orden de los pedidos que se deben terminar en ese momento es irrelevante. Para los trabajos que queden fuera de este término, solo se debe minimizar la suma de sus tiempos de terminación.

Supongamos que hubiera un mejor calendario, diferente al proporcionado por la regla SPT. Consideremos una par de pedidos (i, j) para los cuales $p_i > p_j$ y $C_i < C_j$ "Orden incorrecto"). Si se intercambian estos trabajos, se obtienen nuevos tiempos de terminación, digamos C_i^{nuevo} y C_j^{nuevo}. Los tiempos de terminación de los trabajos anteriores permanecen obviamente sin cambio.

Claramente se tiene $C_i^{nuevo} = C_j$, de manera que tampoco cambian los tiempos de terminación de los trabajos siguientes. Más aún, se cumple $C_j^{nuevo} < C_i$. En caso de que $C_j > d$, se disminuye el valor de la función objetivo, si no es así, entonces permanece constante. Mediante intercambios sucesivos de pedidos contiguos de la manera indicada, se llega al calendario dado por la regla SPT, sin empeorar el valor de la función objetivo. Lo cual es una contradicción a la suposición de que el calendario original es mejor que el dado por la regla SPT. ∎

Por último, falta considerar como caso estándar de los modelos de una máquina el problema $(1|d_j = d|\sum w_j T_j)$. Cuya demostración se puede encontrar en la literatura.

Teorema 4.12. *El problema* $(1|d_j = d|\sum w_j T_j)$ *es NP-difícil.*

Demostración. Véase Yuan [31]. ∎

En opinión de los expertos, parece improbable que exista un procedimiento de orden polinomial para este problema.

Los resultados de este capítulo están resumidos en la siguiente tabla. Los cuales ya se han analizado en este capítulo o como una consecuencia del Corolario 3.31.

		Generalización				**Caso especial**	
$\gamma \diagdown \beta$		r_j	$prec$	s_{jk}	$r_j, pmtn$	$p_j = p$	$d_j = d$
C_{max}	**P**	**P**	**P**	**NP**-dif.	**P**	**P**	**P**
L_{max}	**P**	**NP**-dif.	**P**	**NP**-dif.	**P**	**P**	**P**
$\sum C_j$	**P**	**NP**-dif.	**NP**-dif.	**NP**-dif.	**P**	**P**	**P**
$\sum w_j C_j$	**P**	**NP**-dif.	**NP**-dif.	**NP**-dif.	**NP**-dif.	**P**	**P**
$\sum U_j$	**P**	**NP**-dif.	**NP**-dif.	**NP**-dif.	**P**	**P**	**P**
$\sum w_j U_j$	**NP**-dif.	todos **NP**-difícil				**P**	**NP**-dif.
$\sum T_j$	**NP**-dif.					**P**	**P**
$\sum w_j T_j$	**NP**-dif.					**P**	**NP**-dif.

Tabla 4.1: Resumen de complejidad de los problemas de calendarizacion de una máquina determinista.

Antes de terminar este trabajo relativo a la complejidad, haremos algunas observaciones respecto al método de generalizar o reducir a un problema particular.

Observaciones:

1. Se debe notar, que el caso de interrupción *pmtn* (recíprocamente $r_j, pmtn$) en general no se puede considerar ni como generalización ni como caso especial. A pesar de que aquí hemos tratado esencialmente casos, en los cuales al agregar *pmtn* el problema se simplifica, pero no siempre es el caso. En esta situación se deben hacer algunas observaciones. El valor de la función objeivo no empeora agregando interrupciones, pues siempre se puede establecer un calendario sin interrupciones. Lo peor que puede pasar es que en un caso con interrupciones la función objetivo se vuelva **NP**.

2. El caso de una sola máquina es un caso especial de las otras propiedades de las máquinas mencionadas en este trabajo, tales como Pm, Fm, Jm y Om, así que un problema que sea **NP**-difícil en una sola máquina, también lo es para Pm, Fm, Jm y Om.

4.2. Modelo de dos máquinas en paralelo

Recordando el problema de partición general definido en el capítulo anterior, que establecía: ¿Dado un conjunto de k enteros no negativos $a_1, \ldots, a_k$, existe un subconjunto $I \subseteq \{1, \ldots, k\}$ tal que $\sum_{j \in I} a_j = \sum_{j \in I^c} a_j$?

Para este problema podemos elegir en particular al conjunto $\{6,7,7,2,3,3,3,10,8,15\}$. Si pensamos los elementos del conjunto anterior como los tiempos de procesamiento de 10 trabajos tal como aparecen ordenados, obtendremos la siguiente tabla.

j	1	2	3	4	5	6	7	8	9	10
p_j	6	7	7	2	3	2	2	10	8	15

Utilizando dos máquinas idénticas en paralelo para procesar los trabajos anteriores y considerando como objetivo a minimizar el intervalo de tiempo en el que se procesa completamente la totalidad de los trabajos, es decir, $C_{\text{máx}}$ (makespan) obtenemos el calendario de la figura 4.4.

En este caso, el tiempo total de procesamiento en el calendario anterior es igual a 31, lo que significa que cualquiera de las dos secuencias organizadas puede ser elegida como una partición del problema general, es decir, si elegimos la secuencia de trabajos de la máquina 1 o la máquina 2 como una partición del conjunto $\{6,7,7,2,3,3,3,10,8,15\}$, en ambos casos se cumple la condición del problema de partición general. El siguiente teorema establece la relación entre el problema de partición y el de calendarización $P_2||C_{max}$.

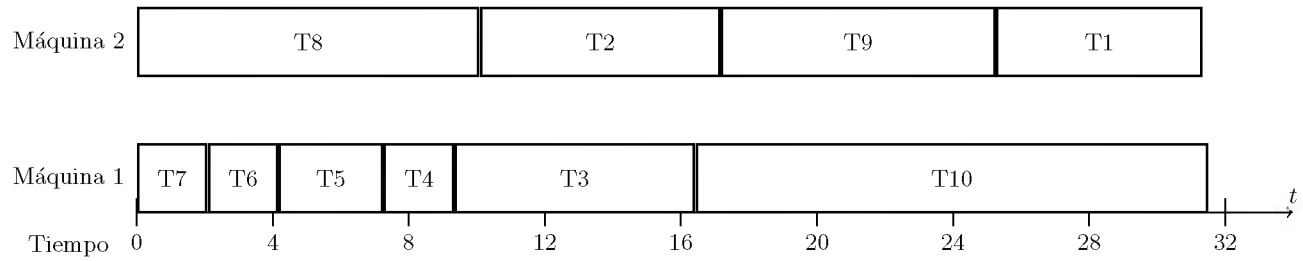


Figura 4.4: Calendario óptimo para dos máquinas.

Teorema 4.13. *El problema general de partición se reduce a* $(\mathbf{P_2}||\mathbf{C_{max}})$.

Demostración. Sea $(a_1,\ldots,a_n)$ una instancia del problema de partición. Para crear una instancia del problema $(\mathbf{P_2}||\mathbf{C_{max}})$ debemos considerar el número de trabajos n, donde a cada trabajo j se le asigna un tiempo de procesamiento a_j. Dado que la suma

$$\sum_{j\in A} a_j = 2\sum_{j\in I} a_j,$$

para alguna partición I de A, es claro que dicha asignación se realiza en tiempo polinomial y es facil ver que existe una partición de A si y solo sí hay un calendario con el tiempo de procesamiento total no mayor a $\frac{1}{2}\sum_{j\in A} a_j$. ∎

En la sección 3.1.3 se habló acerca del modelo no determinista de varias cintas y especialmente en la figura 3.6 se presentó la máquina de tres cintas no determinista que verifica en tiempo polinomial una instancia del problema de partición. Con los teoremas 3.1 y 3.2 es fácil ver que el problema de partición en su versión determinista es un problema intractable acotado exponencialmente.

Aunque el problema de partición puede ser reducido a $P_2||C_{max}$, para este último no existe esperanza alguna de encontrar un algoritmo de tiempo polinomial para resolverlo cuando se consideran tiempos de procesamiento arbitrario, pues el teorema anterior establece que $P_2||C_{max}$ es un problema **NP**-difícil. Sin embargo, al permitir ciertas restricciones al modelo de calendarización anterior es posible calcular fácilmente el C_{max}, en particular si consideramos tiempos de procesamientos idénticos para cada pedido [49].

En particular, el modelo de calendarización de m máquinas idénticas en paralelo $M_1, M_2, \ldots, M_m$ y n trabajos j, tal que $j \in \{1,2,3,\ldots,n\}$ con tiempos de procesamiento p_j que permite asignar un trabajo a varias máquinas sin traslapado (**pmtn**) para el cual se desea minimizar el tiempo de procesamiento de la totalidad de los trabajos (C_{max}) es posible resolverlo polinomialmente mediante la siguiente cota,

$$LB := \max\{\max_j p_j, (\sum_{j=1}^{n} p_j)/m\}.$$

Como un ejemplo sencillo, consideremos $n = 5$ tareas, con los tiempos de procesamiento presentados en la siguiente tabla:

j	1	2	3	4	5
p_j	3	5	7	8	7

La construcción de un calendario optimo se presenta en la figura 4.5, la cual se obtiene de calcular la cota *LB* de los trabajos de la tabla anterior, esto se puede hacer fácilmente como sigue:

$$\begin{aligned} LB &= \max\{\max_j p_j, (\sum_{j=1}^{5} p_j)/3\} \\ &= \max\{8, 10\} \\ &= 10. \end{aligned}$$

En este caso la totalidad de los trabajos se completan en 10 unidades de tiempo, el procedimiento es simple y las implementaciones del modelo pueden realizarse en un tiempo $O(n)$, lo cual es extremadamente rápido.

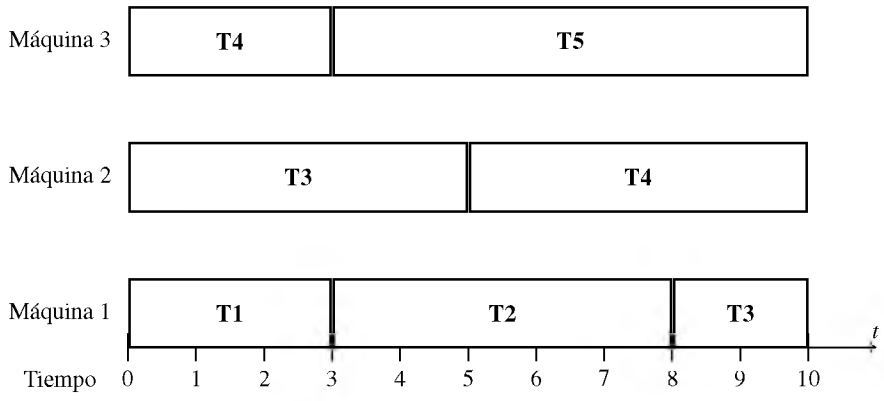


Figura 4.5: Calendario óptimo para una instancia de $P3|pmtn|C_{max}$.

Existen una gran variedad de variantes para el modelo de máquinas en paralelo que se pueden resolver en tiempo polinomial, referente a problemas de máquinas en paralelo dentro de la clase **NP**-completa se puede consultar al artículo de Ullman [18].

Capítulo 5

Conclusiones

Este trabajo fue inspirado en una situación real de máquinas en paralelo, pero debido a la naturaleza aleatoria del problema fue necesario establecer las nociones fundamentales de la calendarización y su relación con la noción de computación y complejidad. Lo anterior nos inspiro a investigar la terminología y métodos básicos de los algoritmos de calendarización de máquinas secuenciales estudiando además la "dificultad" de sus problemas.

Pretendemos que el presente trabajo se pueda usar como referencia para indagar en problemas más generales de calendarización como por ejemplo el problema de piso de trabajo (Job Shop), flujo de taller (Flow Shop), máquinas en paralelo, etc. También deseamos que pueda servir como fuente para abordar de forma general la noción de complejidad de problemas computacionales.

Para concluir, diremos que en este momento contamos con las bases necesarias para atacar el problema mencionado inicialmente, el cual hemos encontrado que se resuelve con la técnica de *programación dinámica estocástica* y requiere también un análisis bastante amplio.

Bibliografía

[1] David Hilbert. Mathematical problems. Lecture delivered before the International Congress of Mathematicians at Paris in 1900. In Mathematical Society, 1976, pp. 1-34. 42

[2] David Hilbert and Wilhelm Ackermann (1928). Grundzüge der theoretischen logik (Principles of Mathematical Logic), Springer-Verlag. 25, 42

[3] Kurt Gödel (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I Monatshefte für Matematik und Physik, 38:173-198. 42

[4] Alonzo Church (1936). Un unsolvable problem of elementary number theory, Amer. J. Math., 58:345-363. 42

[5] Emilio Post (1936). Finite combinatory processes-formulation, I, J. Symbolic Logic, 1:103-105. 42

[6] Alan M. Turing (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, s2-42:230-265. 25

[7] Emilio Post (1943). Formal reductions of the general combinatorial desision problem. Amer. J. Math., 65:197-215. 42

[8] Michael O. Rabin (1960). Degree of dificutly of computing a function and a partial ordering of recursive sets. Technical Report No. 2, Hebrew University, Jerusalem, Israel. 51

[9] J. Hartmanis and R. E. Stearns (1965). On the computational complexity of algorithms. Transactions of the AMS, 117:285-306. 51

[10] Noam Chomsky (1965). Aspects of the Theory of Syntax. Cambridge, MIT Press. 18

[11] Manuel Blum (1967). A machine-independent theory of the complexity of recursive functions, Journal of the ACM, 14(2):322-336 51

[12] Marvin L. Misky (1967). Computation: Finite and infinite machines. Prentice-Hall. 40

[13] Moore J.M. (1968). An n job, one machine scheduling algorithm for minimizing the number of late jobs. Managment science, 15(1):102-109. 5

[14] Lawler E. L. and Moore J. M. (1969). A fuctional equation and its applications to resource allocation and sequencing problems. Managment Science 16(1):77-84. 70

[15] Stephen A. Cook (1971). The Complexity of Theorem-Proving Procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151-158. 61, 65

[16] A. V. Aho, J. E. Hopcroft, J. D. Ullman (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass. 38

[17] Karp R. (1972). Reducibility among combinatorial problem. In Miller, R. and thetcher J. Editors. Complexity of computer Computations, pp. 85-103, Plenum Press. 70

[18] D. J. Ullman (1975). NP-complete Scheduling Problems. Journal of computer and system sciences 10, pp.384-393. 82

[19] A. H. G. Rinnooy Kan (1976). Machine Scheduling Problems: Classification, complexity and computations. Springer.

[20] Lenstra J., Rinnooy Kan, Brucker P. (1977). Complexity of machine scheduling problems. In Hammer P., Johnson E., Korte B. and Nemhauser G. Editoren, Studies in Integer Programming, volume 1 of Annals of Discrete Mathematics, Seiten 343-362. Elsevier. 72

[21] Lenstra J. K. and Rinnooy Kan, A.H.G. (1978). Complexity of scheduling under precedence constraints. Operations Research 26(1):22-35.

[22] Michael R. Garey, David S. Johnson (1979). Computer and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman and Company.

[23] John E. Hopcroft, Jeffrey D. Ullman (1979). Introduction to Automata Theory Languages, and Computation. Addison-Wesley. 41

[24] Graham R. L., Lawler E.L., Lenstra J.K and Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics, 4: 287-326. 8

[25] Lenstra J.K. and Rinnoy Kan, A.H.G. (1980). Complexity results for scheduling chains on a single machine. European Journal of Operational Research 4(4):270-275. 72, 76

[26] Manin, Yu. I. (1981). Lo demostrable e indemostrable, Mir, Moscú. 17

[27] Labetoulle J., Lawler E. L., Lenstra J. K. and Rinnoy Kan, A.H.G. (1982). Preemptive Scheduling of uniform machine subject to release dates, Progress in combinatorial optimization, pp. 245-261. 74

[28] Lawler E. (1990). A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Annals of Operations Research 26: 125-133. 77

[29] J. Du and J.Y.-T. Leung (1990). Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15(3):483–495.

[30] Michael Sipser (1992). The History and Status of the P versus NP Question. STOC, 603-618. 51

[31] Yuan J. (1992). The NP-hardness of the single machine common due date weigted tardiness problem. Systems Sciences and Mathematical Sciences 5(4):328-333. 79

[32] Lathrop, R. (1994). The protein threading problem with sequence amino acid interaction preferences is NP-complete. Protein Engineering vol. 7, no. 9, pp. 1059-1068. 66

[33] Elliot Mendelson (1997). Introduction to mathematical logic. Fourth edition, Chapman & Hall.

[34] Yuri V. Matiyasevich (1997). Hilbet's Tenth Problem, The MIT press, Cambridge, London. 41

[35] Guilles Brassard, Paul Bratley (1999). Fundamentos de algoritmia. Pearson Education.

[36] Martin Davis, Ron Sigal, Elaine J. Weyuker (1994). Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science. Second Edition, Academic Press. 18, 24, 25

[37] Dean Kelley (1995). Teoría de Autómatas y lenguajes formales. Prentice Hall. 17, 25

[38] James L. Hein (1996). Theory of Computation An introduction. Jones and Bartlett Publishers, Inc.

[39] G. Rozenberg, A. Salomaa (eds) (1997). Handbook on Formal Lenguage Volumen 1. Springer-Verlag. 25

[40] Dexter C. Kozen (1997). Automata and Computability. First Edición, Springer. 24, 42

[41] Thomas A. Sudkamp (1997). Lenguages and Machine, An Introduction to the Theory of Computer Science. Secund Edition, Addison-Wesley. 25, 40, 64

[42] Harry R. Lewis, Christos H. Papadimitriou (1998). Elements of the Theory of Computation. Prentice Hall. 42, 66

[43] John E. Savage (1998). Models of Computation, Exploring the Power of Computing. Addison-Wesley. 37

[44] Ding-Zhu Du, Ker-I Ko (2000). Theory of Computational Complexity. John Wiley & Sons, Inc.

[45] Steve Cook (2000). The P vs NP Problem. The Millennium Prize Problems, Clay Mathematics Institute, American Mathematical Society, pp. 87-102. 66

[46] Robert G. Bartle (2000). Introduction Real Analysis. John Wiley & Sons. 15

[47] Stephen Abbott (2001). Understanding analysis. Springer. 17

[48] R. Baron, J. D Durieu, H. Haller and P. Solal (2004). Finding a Nash equilibrium in spatial games is NP-complete. Economic Theory vol. 23, no. 2 , pp. 445-454. 66

[49] Joseph Y-T. Leung (2004). Handbook of SCHEDULING: Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC. 81

[50] Peter Brucker, Sigrid Knust (2006). Complexity Scheduling. Springer-Verlag.

[51] Michael Sipser (2006). Introduction to the Theory of Computation. Secund Edition, Thompson, 2006. 30, 55, 65

[52] Vincent Tkindt, Jean-Charles Billaut (2006). Multicriteria Scheduling: Theory, Models, and Algorithms. Second Edition, Springer-Verlag.

[53] Peter Brucker (2007). Scheduling Algorithms. Fifth Edition, Springer-Verlang.

[54] Jacek Blazewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, Jan Weglarz (2007). Handbook on Scheduling, From Theory to Applications. Springer-Verlag.

[55] Michael Pinedo (2008). Scheduling: Theory, Algorithms, and Systems. Third Edition, Springer.

[56] Conitzer, V. and Sandholm, T (2008). New complexity results about Nash equilibria. Games and Economic Behavior vol. 63, no. 2 , pp. 621–641. 66

[57] Ian Chiswell (2009). A Course in Formal Languages, Automata and Groups. Springer-Verlag.

[58] Kenneth R. Baker, Dan Trietsch (2009). Principles of Sequencing and Scheduling. John Wiley & Sons.

[59] Arindama Singh. Elements of Computation Theory. Springer-Verlag, 2009.

[60] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009). Introduction to Algorithms. Third Edition, MIT Press. 51, 70

[61] John C. Martin (2011). Introduction to Languages and The Theory of Computation. Fourth Edition, McGraw-Hill. 30, 41

[62] Steven Homer, Alan L. Selman (2011). Computability and Complexity Theory. Second Edition, Springer.

[63] Andrew Hodges (2014). Alan Turing: The Enigma. Princeton University press. 25

[64] Dermot Turing (2015). Proof Alan Turing Decoded. The history press. 25

[65] https://www.win.tue.nl/ gwoegi/P-versus-NP.htm 66