UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

IP=PSPACE

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I O

P R E S E N T A:

CARLOS ALBERTO ZAMORA CURA

MEXICO, D. F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CIUDAD UNIVERSITARIA

VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

FACULTAD DE CIENCIAS
División de Estudios Profesionales
Exp. Núm. 55

M. EN C. VIRGINIA ABRIN BATULE
Jefe de la División de Estudios Profesionales
Universidad Nacional Autónoma de México.
P r e s e n t e .

Por medio de la presente, nos permitimos informar a Usted, que habiendo revisado el trabajo de tesis que realizó el pasante Carlos Alberto Zamora Cura con número de cuenta 8720414-0 con el título: IP = PSPACE

Consideramos que reune los méritos necesarios para que pueda continuar el trámite de su Examen Profesional para obtener el título de Actuario.

GRADO NOMBRE Y APELLIDOS COMPLETOS	FIRMA
Dr. Sergio Rajsbaum Gorodezky Director de Tesis	
Dr. Abdón Sánchez Arroyo	
Dr. Hanna Oktaba	
Dr. Javier Bracho Carpizo Suplente	
Dr. Ma. del Carmen López Laiseca Suplente	

Ciudad Universitaria, D.F., a 7 de Junio de 1994

Contenido

Capítulo 1

Introducción.

El haber logrado demostrar que existen problemas que son indecidibles ha sido uno de los logros mas importantes que han tenido las Matemáticas en este siglo. Sin embargo, cuando se ha llegado a que el problema en cuestión es *decidible* entonces puede suceder que éste sea un problema, ya sea, *Tratable* ó *Intratable*. En general, un problema se considera que es Tratable si su *Complejidad de Tiempo* está acotada polinomialmente (esto es porque su requerimiento de tiempo crece "suavemente" con respecto a el tamaño de su entrada) y es Intratable si su Complejidad de Tiempo no está acotada polinomialmente. Los problemas Intratables tienen Complejidad de Tiempo al menos exponencial.

La clase **NP**, que es la clase de los problemas que pueden ser resueltos con algoritmos no deterministas de tiempo polinomial, juega un papel central en las investigaciones sobre la intratabilidad de problemas. Por definición, **NP** contiene a la clase **P**, que es la clase de aquellos problemas que pueden ser resueltos en forma determinista y en tiempo polinomial. Se sabe que todo problema en **NP** puede resolverse en a lo mas tiempo exponencial; esto es, **NP** está contenido en **EXPTIEMPO**, que es la clase de problemas que pueden ser resueltos en tiempo determinista exponencial. De lo anterior se tiene, en resumen, que:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIEMPO}.$$

Se sabe que **P** está contenido propiamente en **EXPTIEMPO** [8]. Sin embargo, no se conoce si **P** está contenido propiamente en **NP**, ni tampoco si **NP** está contenido propiamente en **EXPTIEMPO**. Consecuentemente la brecha entre los problemas tratables (**P**) y los problemas intratables (**EX-**

PTIEMPO) está dada por **NP**.

Muchos problemas muy importantes (tanto teóricos como prácticos) se sabe que se encuentran en **NP**, y que son de lo mas difíciles dentro de la clase (**NP**-completos). Si se llegara a saber que **NP** = **P**, entonces todos estos problemas tendrían una solución tratable.

Se conoce que **NP** está contenido en **PESPACIO** y que **PESPACIO** está contenido en **EXPTIEMPO**, donde **PESPACIO** es la clase de los problemas que se pueden resolver de forma determinista usando espacio polinomial. Estas contensiones entre las clases presentadas sugieren que estudiar a **PESPACIO** es muy parecido a estudiar a **NP** (**NP** $\subseteq$ **PESPACIO** $\subseteq$ **EXPTIEMPO**). En este trabajo se estudia la relación entre **PESPACIO** e **IP**, donde **IP** es la clase de problemas aceptados en tiempo polinomial por *Sistemas Interactivos de Prueba*. La clase **IP** y los *Sistemas Interactivos de Prueba* fueron propuestos independientemente en [17], [1] y [2]. Un problema que dejaron abierto aquellos artículos fué la relación que tenía **IP** con respecto a las otras clases de lenguajes que se encuentran en la vecindad de **NP**.

Lo mejor que se podía decir era que **IP** estaba contenido en **PESPACIO** (lo cual se prueba en [6] y [2]) y que **IP** contenía a **NP**. Se pensaba que la contensión **NP** $\subseteq$ **PESPACIO** podía ser propia, pues existían lenguajes que pertenecían a **IP** y que no se sabía si pertenecían a **NP** (como el problema de las Gráficas No Isomorfas: Dadas dos gráficas G y H ¿ son H y G isomorfas?).

Sin embargo, al probarse en [19] y [18] que **PESPACIO** $\subseteq$ **IP** se llegó a una caracterización mucho mas fina de los lenguajes en **NP**. En el presente trabajo sólo se considera la demostración de que **PESPACIO** $\subseteq$ **IP** puesto que es la que ha tenido consecuencias mayores dentro de Teoría de Complejidad.

La prueba presentada en [19] y [18] se basa en que para dos clases de lenguajes dados $\mathcal{X}$ y $\mathcal{Y}$, para probar que $\mathcal{X}$ contiene a $\mathcal{Y}$ es suficiente probar que un problema completo para la clase $\mathcal{Y}$, puede ser resuelto usando la definición computacional para la clase $\mathcal{X}$. Esta es la idea de la prueba de **PESPACIO** $\subseteq$ **IP**.

El trabajo está dividido en cuatro capítulos. En el primero de ellos se dan algunas definiciones básicas con las que se trabajará durante el resto de la tésis. En el segundo, se muestran algunos resultados acerca de las relaciones entre Complejidad de Espacio y Complejidad de Tiempo. En el tercero, se estudian las ideas de Transformaciones Polinomiales y Clases Completas para poder llegar a que *QBF* está en **PESPACIO**-completo; también se definen los Sistemas Interactivos de Prueba y se demuestra que

para el problema de las Gráficas No Isomorfas existe un Sistema Interactivo de Prueba. El último capítulo se encarga de demostrar que **PESPACIO** $\subseteq$ **IP**; el Sistema Interactivo de Prueba presentado está basado en el sistema propuesto en [19].

Capítulo 2

Preliminares

En el presente capítulo haremos un breve repaso de las nociones y terminología básicas en Teoría de Lenguajes, así como de las definiciones formales acerca de los modelos de cómputo determinista y no determinista.

2.1 Alfabetos, Palabras y Lenguajes.

Definición 1

1. *Un* alfabeto *es cualquier conjunto no vacío.*
2. *Un* símbolo *es un elemento de un alfabeto.*
3. *Dado un alfabeto* Σ, *una* palabra *es una sucesión finita de símbolos de* Σ.
4. *La* longitud *de una palabra* ω, *denotada por* $\mid \omega \mid$, *es el número de símbolos de* ω.
5. *La* palabra vacía *(o* nula*) denotada por* ϵ, *es la palabra que consiste de cero símbolos.*

Dado un alfabeto Σ, denotamos como Σ^n a el conjunto de todas las palabras formadas por símbolos de Σ con longitud $n \in \mathbf{N}$. Cuando $n = 0$ definimos a $\Sigma^0 = \{\epsilon\}$. Definimos también a:

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n.$$

Definición 2 (Lenguajes) .
Dado un alfabeto Σ, *un* Lenguaje *sobre* Σ *es un subconjunto de* Σ^*.

2.2 Máquinas de Turing.

La Máquina de Turing es un dispositivo abstracto que permite, mediante operaciones elementales (lectura y escritura), realizar el reconocimiento de Lenguajes. La podemos conceptualizar intuitivamente como un dispositivo, llamado *control finito*, el cual lee mediente cabezas lectoras un símbolo de cada una de sus k cintas infinitas, las cuales están divididas en celdas, después de leer los símbolos escribe sobre cada una de sus cintas un símbolo y, sobre cada cinta, mueve su cabeza lectora una celda a la izquierda (L), una a la derecha (R) ó no mueve la cabeza lectora (N). La forma de decidir acerca de estas operaciones está dada por su función de transición (de aquí que podamos conceptualizar a la función de trancisión como el Control Finito).

Definición 3 (Máquina de Turing) .
Una Máquina de Turing Determinista de k cintas en línea *MTD es un 5-tuplo*

$$M = \langle Q, \Sigma, \delta, q_0, F, b \rangle$$

donde

1. Q *es un conjunto finito de los estados internos.*
2. Σ *es el alfabeto de las cinta.*
3. $q_0 \in Q$ *es el estado inicial de* M.
4. $F \subseteq Q$ *es el conjunto de estados de aceptar.*
5. $\delta : Q \times \overbrace{\Sigma \times \cdots \times \Sigma}^{k \text{ veces}} \longrightarrow \overbrace{\Sigma \times \cdots \times \Sigma}^{k \text{ veces}} \times Q \times \overbrace{\{R, L, N\} \times \cdots \times \{R, L, N\}}^{k \text{ veces}}$
   es una función parcial llamada "función de transición" de M.
6. $b \in \Sigma$, *símbolo especial llamado el* símbolo blanco *de* M.

Una *MTD* comienza a operar teniendo en alguna de sus cintas una palabra de entrada ω (la cinta sobre la cual aparecerá ω puede tomarse siempre como la misma para facilitar el diseño de la función de transición) y en

las demás celdas (sobre todas la cintas) el símbolo blanco. La función de transición comienza a operar teniendo como estado inicial a q_0. La *MTD* procede a aplicar la función de trancisión según lo marque el contenido de sus cintas. Puesto que δ es una función parcial entonces puede suceder que no esté definida para una posible entrada, en este caso se dice que la máquina *para* (pues no puede seguir realizando transiciones). Un *paso* es una aplicación de la función de transición. Si después de una sucesión de pasos de la Máquina de Turing para en algún estado de F (aceptar), entonces se dice que *acepta* a la palabra de entrada; en caso contrario se dice que la máquina *rechaza* a su palabra de entrada. Si no damos alguna restricción acerca de la forma y cantidad de pasos que puede realizar δ, entonces podemos conceptualizar este hecho como que las cintas de la Máquina de Turing son *infinitas* (a ambos lados). Si por el contrario, restringimos el funcionamiento de la función de transición a no ir mas allá de cierta celda, entonces podemos decir que las cintas de la máquina son semi-infinitas (es decir, infinitas sólo hacia la izquierda ó a la derecha).

Definición 4 *Dada una Máquina de Turing M, una* configuración *de M, también llamada* descripción instantánea, *es una descripción del estado completo de la Máquina: incluye el contenido de cada una de sus cintas, la posición de cada una de las cabezas lectoras y el estado de el control finito. Si M tiene k cintas, una configuración de M es:*

$$(q, x_1, \ldots, x_k)$$

donde q es el estado de M y cada $x_j \in \{\omega \in \Sigma \cup \{\#\} \mid \omega = x\#y \text{ con } x, y \in \Sigma^*, \# \notin \Sigma\}$, el cual, representa el contenido actual de la j-ésima cinta. El símbolo $\#$ marca la posición de la cabeza lectora de M para cada una de sus cintas.

Con las definiciones anteriores podemos definir el *Lenguaje aceptado por una Máquina de Turing.*

Definición 5 *Sea $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ una Máquina de Turing Determinista, entonces el* Lenguaje que acepta esta máquina *es:*

$$L_M = \{\omega \in \Sigma^* \textit{ tal que } M \textit{ acepta a } \omega\}.$$

Dado el funcionamiento de una MTD, se tiene que para una entrada ω, la máquina puede ó no aceptar a ω. En caso de que no la acepte puede

seceder que: la máquina pare en un estado que no pertenece a F ó bien que la máquina nunca pare. El hecho de que para un lenguaje L exista una Máquina de Turing la cual pare sobre todas las palabras de L (aceptando o rechazando) es muy importante, pues se puede siempre decidir cuando una palabra está o no en L. Por estas razones, esta clase de lenguajes recibe una denominación especial.

Definición 6 *Un lenguaje L sobre Σ, se dice que es* recursivo *si existe una Máquina de Turing que acepte a L y que pare para todas las entradas en $\Sigma^* \setminus \{\epsilon\}$.*

2.3 No Determinismo

Dada una *Máquina de Turing Determinista* y una palabra de entrada, todo movimiento de la máquina está completamente determinado. El estado de la Máquina y los símbolos revisados en cualquier momento, determinan completamente su siguiente estado, movimiento y símbolo a escribir. Una *Máquina de Turing no Determinista MTND* está definida como un 6-tuplo:

$$M = \langle Q, \Sigma, \delta, q_0, F, b \rangle$$

donde cada elemento se define de igual manera que en el caso determinista, con excepción de que

$$\delta : Q \times \overbrace{\Sigma \times \cdots \times \Sigma}^{k \text{ veces}} \longrightarrow \mathcal{P}(\overbrace{\Sigma \times \cdots \times \Sigma}^{k \text{ veces}} \times Q \times \overbrace{\{R, N, L\} \times \cdots \times \{R, N, L\}}^{k \text{ veces}})$$

donde $\mathcal{P}(\cdot)$ denota la función potencia de un conjunto.

Todas las definiciones hechas hasta el momento para el caso determinista se mantienen para el caso no determinista. Sin embargo, dada una entrada ω para una *Máquina de Turing No Determinista M* es posible que no exista sólo una forma de que M actúe sobre ω, por lo tanto, la manera en como se puede computar una palabra de entrada puede ser distinto.

Definición 7 *Una palabra ω es* aceptada *por una Máquina de Turing No Determinista si existe al menos alguna manera de que M actúe sobre ω y termine en algún estado de aceptación.*

El hecho de menejar no determinismo no implica que tengamos mas capacidad de cómputo, es decir, la clase de lengiajes que se pueden aceptar en *MTND* es igual a la clase de lenguajes que puden ser aceptados en *Máquina de Turing Determinista* (para una demostración de este hecho ver [10]).

Capítulo 3

Complejidad Computacional.

Una posible clasificación de los Lenguajes Formales, en Teoría de Computación, está basada en la cantidad de recursos, por ejemplo espacio o tiempo, para reconocerlos en algún modelo universal de cómputo. La Teoría de Complejidad Computacional se encarga de crear y estudiar los algoritmos para reconocer a los lenguajes, así como de "medir" la cantidad de recursos (en algún modelo de cómputo) que requieren estos algoritmos. En el presente capítulo se dará una introducción a la Teoría clásica de Complejidad Computacional, presentando sus definiciones y resultados básicos. Este capítulo esta basado, principalmente en las exposición que se hace acerca de estos temas en [10] y [12].

3.1 Definiciones y primeros resultados

Como en la teoría clásica, utilizaremos la máquina de Turing como modelo universal de cómputo; consideremos la Máquina de Turing *fuera de línea* con k cintas.

3.1.1 Complejidad de Espacio.

Utillizaremos la Máquina de Turing (MT) como modelo de cómputo, definiremos la *Complejidad Espacio* de lenguajes sólo sobre lenguajes recursivos y a partir de su Máquina de Turing asociada. En particular usaremos el siguiente modelo de Máquina de Turing de k-cintas.

El hecho de menejar no determinismo no implica que tengamos mas capacidad de cómputo, es decir, la clase de lengiajes que se pueden aceptar en *MTND* es igual a la clase de lenguajes que puden ser aceptados en *Máquina de Turing Determinista* (para una demostración de este hecho ver [10]).

Capítulo 3

Complejidad Computacional.

Una posible clasificación de los Lenguajes Formales, en Teoría de Computación, está basada en la cantidad de recursos, por ejemplo espacio o tiempo, para reconocerlos en algún modelo universal de cómputo. La Teoría de Complejidad Computacional se encarga de crear y estudiar los algoritmos para reconocer a los lenguajes, así como de "medir" la cantidad de recursos (en algún modelo de cómputo) que requieren estos algoritmos. En el presente capítulo se dará una introducción a la Teoría clásica de Complejidad Computacional, presentando sus definiciones y resultados básicos. Este capítulo esta basado, principalmente en las exposición que se hace acerca de estos temas en [10] y [12].

3.1 Definiciones y primeros resultados

Como en la teoría clásica, utilizaremos la máquina de Turing como modelo universal de cómputo; consideremos la Máquina de Turing *fuera de línea* con k cintas.

3.1.1 Complejidad de Espacio.

Utillizaremos la Máquina de Turing (MT) como modelo de cómputo, definiremos la *Complejidad Espacio* de lenguajes sólo sobre lenguajes recursivos y a partir de su Máquina de Turing asociada. En particular usaremos el siguiente modelo de Máquina de Turing de k-cintas.

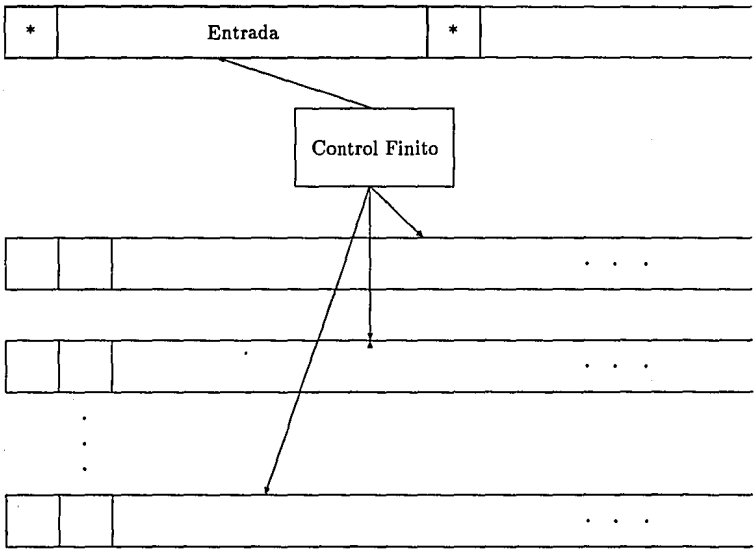


Fig. 1.1 TM multicintas con cinta de solo-lectura.

Definición 8 (MT Determinista de k-cintas semi-infinitas fuera de línea) *Una Máquina de Turing M, de k-cintas semi-infinitas determinista fuera de línea es una Máquina de Turing de k cintas, con $k > 1$, tal que la primera cinta es considerada como una cinta de entrada de sólo-lectura. Sobre esta cinta, la palabra de entrada esta delimitada por símbolos *, tal que * no está en el alfabeto de cinta de M. También consideramos que todas las cintas de M son semi-infinitas a la derecha (esta consideración se hace dentro de la función de trancisión de M). fig 1.1.*

En todas las definiciones y resultados concernientes a Complejidad Espacio se utilizará una MT (Máquina de Turing) fuera de línea con una cinta

de sólo-lectura con marcas finales y k cintas de trabajo semi-infinitas. (Fig. 1.1)

Definición 9 (Complejidad Espacio) .
Sea M una MT determinsta fuera de línea con k cintas de trabajo semi-infinitas con alfabeto Σ. Si $\forall\omega \in \Sigma^*$ tal que $\mid \omega \mid = n$ con ω palabra de entrada para M, la máquina examina a lo más $S(n)$ celdas sobre cualquier cinta entonces decimos que M es una MT de Complejidad de Espacio S(n). *Un lenguaje L es de Complejidad Espacio $S(n)$, si existe una Máquina de Turing M, tal que M acepta a L y M es de Complejidad Espacio $S(n)$.*

Por ser M una MT fuera de línea se tiene que su Complejidad de Espacio siempre es mayor a la longitud de la palabra de entrada (por las condiciones sobre su cinta de entrada).

3.1.2 Complejidad de Tiempo.

Para todos los resultados concernientes a la Complejidad de Tiempo utilizaremos una Máquina de Turing determinista de k cintas infinitas en línea, con $k \geq 1$, donde una de las cintas contiene la palabra de entrada. Notemos que para Complejidad de Tiempo no utilizaremos Máquinas de Turing fuera de línea. fig 1.2.

Definición 10 (Complejidad de Tiempo) .
Sea M una MT determinista de k cintas infinitas con alfabeto Σ. Si $\forall\omega \in \Sigma^*$ con $\mid \omega \mid = n$ tal que ω es la palabra de entrada para M, sucede que M hace a lo más $T(n)$ pasos antes de parar sobre cada una de sus cintas, entonces decimos que M es una MT de Complejidad de Tiempo T(n). *Un Lenguaje L es de Complejidad de Tiempo $T(n)$ si existe una Máquina de Turing M, tal que M acepta L y M es de Complejidad de Tiempo $T(n)$.*

Para la funciones de complejidad de tiempo y espacio podemos hacer ciertos supuestos:
Nuestro supuesto para las funciones de Complejidad de Espacio :

> Cuando hablamos de Complejidad Espacio $S(n)$ en realidad hablamos de $max\{1, S(n)\}$.

Esto es natural, pensando que toda MT utiliza al menos una celda para todas las entradas.
Nuestro supuesto para las funciones de Complejidad Tiempo :

Cuando hablamos de Complejidad Tiempo $T(n)$ en realidad hablamos de $max\{n+1, T(n)\}$.

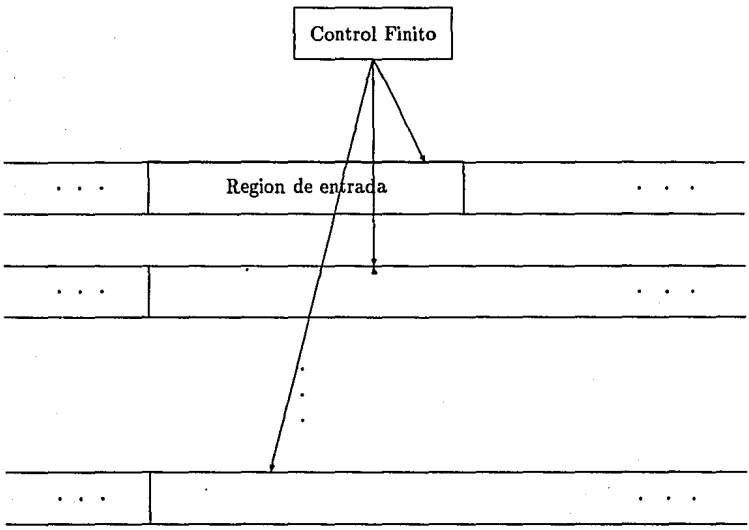


Fig. 1.2 TM multicintas.

Podemos justificar este supuesto pensando que cualquier función de Complejidad de Tiempo $T(n)$ lee la entrada completa hasta alcanzar el primer símbolo blanco. Existen lenguajes en los cuales se pueden construir MT en las cuales no sea necesario leer toda la entrada. Pensemos en $L = \{\omega \mid \omega = 0\alpha con \alpha \in \{0,1\}^*\}$, para este lenguaje podemos construir una MT la cual acepta a L y no necesita leer toda la entrada. Este tipo de lenguajes los eliminamos de nuestras consideraciones, pensando que si no necesitan leer toda la entrada para decidir su aceptación, entonces estos lenguajes son "fáciles" de reconocer y no necesitan grandes capacidades de cómputo para su reconocimiento.

Complejidad de Tiempo y de Espacio No Determinista

Podemos generalizar los conceptos que hasta ahora hemos definido en MT deterministas para MT No Deterministas. Una MT no determinista es de Complejidad Tiempo $T(n)$ si no existe alguna sucesión de movimientos que hagan que la máquina ejecute más de $T(n)$ movimientos, para cualquier entrada de longitud n. Una MT no determinista es de Complejidad de Espacio $S(n)$ si no existe alguna sucesión de movimientos que permitan que la máquina examine más de $S(n)$ celdas sobre cualquiera de sus cintas de trabajo y para cualquier entrada de longitud n.

3.1.3 Clases de Complejidad

A partir de los conceptos hasta ahora presentados podemos definir familias de lenguajes que llamaremos *Clases de Complejidad.*

$$
\begin{aligned}
\mathbf{DESPACIO(S(n))} &= \{L \mid L \textit{ tiene complejidad espacio } S(n)\} \\
\mathbf{NESPACIO(S(n))} &= \{L \mid L \textit{ tiene complejidad} \\
&\qquad \textit{espacio no determinista } S(n)\} \\
\mathbf{DTIEMPO(T(n))} &= \{L \mid L \textit{ tiene complejidad tiempo } T(n)\} \\
\mathbf{NTIEMPO(T(n))} &= \{L \mid L \textit{ tiene complejidad tiempo} \\
&\qquad \textit{no determinista } T(n)\}
\end{aligned}
$$

Estas clases de lenguajes que acabamos de definir son básicas en Teoría de Computación, pués a partir de estas se construyen clases de complejidad específicas, las cuales tienen gran importancia tanto teórica como práctica. Por ejemplo, podemos citar las siguientes:

$$
\begin{aligned}
\mathbf{LOG} &= \bigcup_{c\geq 1} \mathbf{DESPACIO(c \log_2 n)} \\
\mathbf{NLOG} &= \bigcup_{c\geq 1} \mathbf{NESPACIO(c \log_2 n)} \\
\mathbf{P} &= \bigcup_{i\geq 0} \mathbf{DTIEMPO(n^i)} \\
\mathbf{NP} &= \bigcup_{i\geq 0} \mathbf{NTIEMPO(n^i)} \\
\mathbf{PESPACIO} &= \bigcup_{i\geq 0} \mathbf{DESPACIO(n^i)} \\
\mathbf{NPESPACIO} &= \bigcup_{i\geq 0} \mathbf{NESPACIO(n^i)}
\end{aligned}
$$

$$\begin{aligned}
\mathbf{DEXP} &= \bigcup_{c\geq 0}\mathbf{DTIEMPO}(2^{\mathbf{cn}}) \\
\mathbf{NEXP} &= \bigcup_{c\geq 0}\mathbf{NTIEMPO}(2^{\mathbf{cn}}) \\
\mathbf{EXPESPACIO} &= \bigcup_{c\geq 0}\mathbf{DESPACIO}(2^{\mathbf{cn}}) \\
\mathbf{EXPTIEMPO} &= \bigcup_{c\geq 0}\mathbf{DTIEMPO}(2^{\mathbf{n^c}}) \\
\mathbf{NEXPTIEMPO} &= \bigcup_{c\geq 0}\mathbf{NTIEMPO}(2^{\mathbf{n^c}})
\end{aligned}$$

De aquí la importancia de conocer las propiedades y relaciones básicas las clases de complejidad **DTIEMPO, NTIEMPO, DESPACIO** y **NPESPACIO** para comprender mejor la importancia del teorema objetivo de esta tésis.

3.2 Aceleración, Compresión y Reducción en cintas.

Puesto que el tamaño del control finito de una *MT* (número de estados y tamaño de su alfabeto) puede ser arbitrariamente grande, entonces es posible que el espacio y el tiempo necesario para reconocer un lenguaje pueda ser escalado siempre por un factor constante. Esto tiene gran importancia pues nos lleva a considerar que los factores constantes pueden ser ignorados. En la presente sección presentaremos algunos hechos básicos al respecto.

3.2.1 Compresión de cinta.

Teorema 1 (Teorema de Compresión de Cinta) .
Si L es aceptado por una MT con k cintas de trabajo y complejidad de espacio acotado por $S(n)$ entonces $\forall c > 0$, L es aceptado por una MT con complejidad de espacio $cS(n)$.

Demostración :
Sea M_1 una MT fuera de línea de complejidad de espacio S(n) y tal que $L_{M_1} = L$. La idea es construir una MT M_2 que simule a M_1 en donde para alguna constante entera positiva r, cada celda en las cintas de trabajo de M_2 represente el contenido de r celdas adyacentes de M_1.

Sea $r \in \mathbf{N}$ tal que $rc \geq 2$.

Cada símbolo del alfabeto de cinta de M_2 será un r-tuplo de símbolos del alfabeto de M_1. Los estados de M_2 son tuplos de la forma $(q, i_1, \ldots, i_k)$ donde

q es un estado de M_1 y $1 \leq i_m \leq r$, $m \in \{1, \cdots, k\}$. Cada configuración de M_2 corresponderá a una configuración de M_1 de la siguiente manera: el j-ésimo símbolo de cada cinta de M_2 codifica los símbolos en las celdas $(j-1)r+1$ al jr de la cinta correspondiente en M_1. Para cada cinta $m, 1 \leq m \leq k$, cuando la cabeza lectora de M_1 examina al símbolo $(j-1)r+l$ y está en el estado q, M_2 se encuentra en el estado $(q, \cdot, \ldots, l, \ldots, \cdot)$. De esta manera en cada configuración, M_2 conoce el símbolo examinado por cada una de las cabezas lectoras de M_1 y por lo tanto es posible simular correctamente el comportamiento de M_1 actualizando los contenidos de las cintas y los estados.

Puesto que tomamos grupos de r celdas en M_2, entonces M_2 puede simular a M_1 en no más de $\lceil \frac{S(n)}{r} \rceil$ celdas sobre cualquier cinta de trabajo.

Si $c \geq 1$ entonces $L_{M_1} \in$ **DESPACIO(c(S(n))** pues $S(n) \leq cS(n)$.

Consideremos $c < 1$, como $rc \geq 2$ entonces $\frac{2}{r} \leq c$, de aquí que si $r \leq S(n)$, entonces:

$$\frac{S(n)}{r} \leq 2\frac{S(n)}{r} \leq cS(n)$$

como $r \leq S(n)$

$$\lceil \frac{S(n)}{r} \rceil \leq \frac{S(n)}{r} + 1 \leq \frac{S(n)}{r} + \frac{S(n)}{r} = 2\frac{S(n)}{r} \leq cS(n)$$

y por lo tanto

$$L_{M_2} \in \textbf{DESPACIO(cS(n))}$$

Si $r > S(n)$ entonces M_2 utiliza una celda en el peor de los casos y como $\frac{S(n)}{r} < 1$ entonces $\lceil \frac{S(n)}{r} \rceil = 1$ y por lo tanto

$$L_{M_2} \in \textbf{DESPACIO(cS(n))}$$

(Pues la complejidad de espacio es $max\{1, \lceil cS(n) \rceil\}$). ■

De el teorema anterior podemos obtener el siguiente:

Corolario 1
Si $L \in$ **NESPACIO(S(n))** *entonces* $\forall c > 0$ $L \in$ **NESPACIO(cS(n))**.

Notemos que según la construcción presentada el costo de reducir el espacio de cinta utilizado por una MT es aumentar tanto el tamaño del control finito como de su alfabeto.

Presentemos ahora el siguiente teorema clásico, que nos afirma que el número de cintas necesarias para reconocer algún lenguaje pueden reducirse.

Teorema 2 *Sea $L = L_{M_1}$ tal que M_1 es una MT de espacio acotado por $S(n)$ con k cintas (fuera de línea) entonces L es aceptado por una MT con una sola cinta (fuera de línea) y de espacio acotado por $S(n)$.*

Demostración :

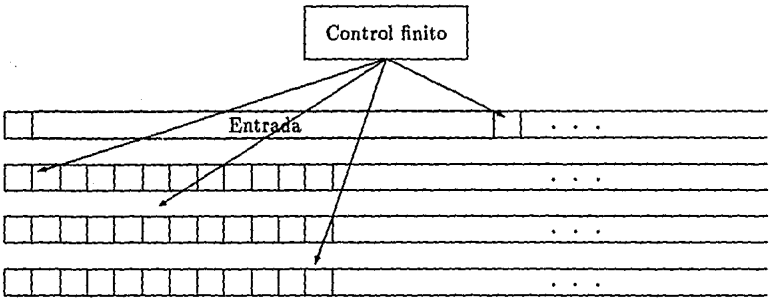
Sea L tal que $L = L_{M_1}$ para alguna Máquina de Turing M_1 fuera de línea de espacio acotado por $S(n)$ con k cintas. Debemos construir una Máquina de Turing M_2 fuera de línea con una sola cinta tal que $L = L_{M_2}$ y M_2 sea de espacio acotado $S(n)$.

Construiremos M_2 con una sola cinta y $2k$ pistas, dos pistas para cada cinta de trabajo de M_1. Una de sus pistas grabará el contenido de su correspondiente cinta en M_1 y la segunda estará en blancos, excepto por una *marca* en la celda que está siendo examinada por la correspondiente cabeza de M_1. (Fig. 1.3)

El control finito de M_2 almacena el estado de M_1 y un *contador* de el número de marcas (en la cinta de trabajo de M_2) que se encuentran a la derecha de la cabeza en la cinta de trabajo de M_2.

Inicialmente la cabeza de M_2 estará en la celda marcada que se encuentra más a la derecha de la cinta. Cada movimiento de M_1 se simula por dos *barridas*. La primera se hace de derecha a izquierda visitando cada una de las celdas con marcas y grabando el símbolo examinado por cada cabeza de M_1; cuando M_2 cruza una marca actualiza el contador de marcas, cuando ya no hay más marcas a su derecha (es decir cuando el marcador es igual a k) M_2 ha visitado todos los símbolos examinados por las cabezas de M_1. La segunda barrida se realiza de izquierda a derecha, en esta, M_2 alcanza la marca más a izquierda de la cinta (se coloca en posición para simular otro movimiento de M_1), decrementa su contador cada vez que pasa por una marca, actualiza el símbolo de cinta que M_1 está examinando (en la celda de la marca) y mueve la marca de cabeza un símbolo a la izquierda o a la derecha dependiendo del movimiento realizado por M_1. Finalmente, para completar un movimiento de M_1, M_2 cambia el estado de su control al correspondiente de M_1. Si el nuevo estado de M_1 acepta entonces M_2 acepta.

M1 :



M2 :

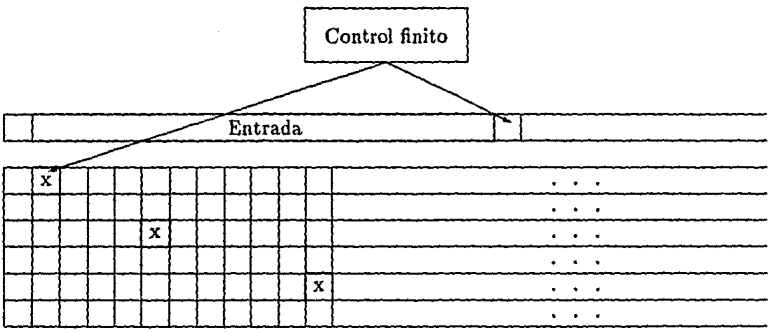


fig. 1.3 Ejemplo del teorema 2 con una TM de 3 cintas.

Ahora bien, por la construcción de M_2, si para todas la cintas de M_1, estas no ocupan más de $S(n)$ celdas (para una entrada ω de tamaño n), entonces como M_2 simula todas éstas en pistas de sus cintas y estas pistas contienen la misma información que sus correspondientes cintas en M_1, entonces M_2 no ocupa más de $S(n)$ celdas para una entrada de tamaño n.

■

Si observamos con cuidado el Teorema 1 podemos concluir que no hay necesidad de simular cada paso de M_1 al interior de un paso en M_2. Algunos pasos pueden ser simulados al interior de uno, con tal que la cabeza de la cinta permanezca "cerca". Por ejemplo, si M_1 opera t pasos en una región de la cinta, de la cual conocemos su contenido antes de que comienze a operar y donde esa región de la cinta de M_1 sea representado por un solo símbolo en M_2, podemos es posible entonces diseñar la función de transición de M_2 para que opere en un solo paso. Esta observación nos sirve para probar el siguiente teorema:

Teorema 3 *Sea L un lenguaje reconocido por una Máquina de Turing M_1 de k cintas fuera de línea cuya Complejidad de Tiempo está acotada por $T(n)$, entonces existe una Máquina de Turing M_2 de k cintas, fuera de línea de Complejidad de Tiempo acotado por $cT(n)$ $\forall c > 0$ con tal que $k > 1$ y*

$$\inf_{n\to\infty} \frac{T(n)}{n} = \infty.$$

Para demostrar este teorema primero demostraremos el siguiente :

Lema 1 *Sea L que es aceptado por alguna MT M_1 de k cintas (fuera de línea) $k > 1$ y $r \in \mathbf{N}$ entonces existe una Máquina de Turing M_2 tal que $\forall \omega \in L$ con $|\omega| = n$ M_1 acepta a ω en $n + \lceil \frac{n}{r} \rceil + 6\lceil \frac{t}{r} \rceil$ pasos, si M_1 acepta a ω en t pasos.*

Demostración :

Construiremos M_2 de manera similar a como se construyó en el Teorema de Compresión. Cada símbolo del alfabeto de M_2 será un r-étuplo (el cual llamaremos *bloque*) de símbolos del alfabeto de M_1 y algunos de los estados de M_2 se codificarán como tuplos $(q, i_1, \cdots, i_k)$ donde q es un estado de M_1 y $1 \leq i_m \leq r$ $m \in \{1, \cdots, k\}$. (Aquí surge una diferencia con el Teorema de Compresión, pues M_2 necesitará más estados para realizar otros cómputos).

Los pasos de M_2 serán reunidos en grupos de 6 pasos, cada grupo de 6 pasos lo llamaremos *Paso básico.* M_2 simulará exactamente r pasos de M_1 en cada paso básico. Un paso básico consta de tres fases que se realizan simultáneamente en cada cinta.

En la primera fase, M_2 usa cuatro pasos para grabar en su control finito los contenidos de las celdas adyacentes a la celda examinada actualmente por sus cabezas de cinta. Esto lo realiza moviendo sus cabezas de cinta un paso a la izquierda, dos a la derecha y uno a la izquierda. Al finalizar estos movimientos M_2 ha vuelto a sus celdas iniciales y tiene la información de $3r$ símbolos de cinta de M_1, por cada cinta de M_1.

En la segunda fase, M_2 encuentra en un paso los contenidos de estas $3r$ celdas de cinta de M_1, después de r movimientos. Esta es una cantidad finita de información la cual puede ser implementada en la función de trancisión de M_2 (en algún sentido, podríamos pensar que esta información la estamos implementando en hadware). Además, este paso puede ser implementado con los cuatro pasos de la primera fase al momento de ir leyendo los contenidos de las celdas. Esto es posible por lo siguiente: observamos que en r pasos la cabeza de M_1 puede cambiar símbolos en a lo más 2 bloques adyacentes, ya sea el examinado actualmente y el bloque de la izquierda ó el actual y el de la derecha. No puede escribir en los tres bloques, pues cada uno de los tiene r símbolos de longitud. Así, la función de transición de M_1 indica los nuevos contenidos a imprimir en a lo más 2 de los 3 bloques y el nuevo lugar para la cabeza de la cinta de M_2, el cual es uno de los bloque modificados.

En la tercera fase, M_2 actualiza los contenidos de las cintas en 2 movimientos: uno para actualizar los contenidos de la celda actual y el otro para modificar el contenido del bloque de la izquierda ó derecha. Estos movimientos deben ser realizados en el orden adecuado dependiéndo de la celda examinada por M_1.

En estas tres fases, cada paso básico simula r movimientos de M_1. Hay que notar que para comenzar el cómputo M_2 debe codificar (comprimir) la entrada, esto lo hace leyendo la entrada y copiándola en cualquier cinta de trabajo, codificando r símbolos al interior de uno (sabemos que M_2 tiene k cintas y que $k > 1$).

El cálculo total del tiempo sobre la entrada ω de longitud n es (por construcción):

1. n pasos para comprimir la entrada
2. $\lceil \frac{n}{r} \rceil$ para regresar la cabeza de cinta a la izquierda de la entrada comprimida.
3. $\lceil \frac{t}{r} \rceil$ pasos básicos para simular t pasos de cómputo de M_2

Como cada paso básico requiere de 6 movimientos entonces obtenemos en total :

$$n + \lceil \frac{n}{r} \rceil + 6\lceil \frac{t}{r} \rceil$$

pasos. ■

Con este lema podemos demostrar el teorema 3.

Demostración : (teorema 3)

Sea $r \in \mathbf{N}$ tal que $rc > 12$ y construyamos M_2 como se indica en el lema anterior. Obtenemos que cada palabra de longitud n en L es aceptado por M_2 en $n + \lceil \frac{n}{r} \rceil + 6\lceil \frac{T(n)}{r} \rceil$ pasos. Tenemos que:

$$\begin{aligned} n + \lceil \frac{n}{r} \rceil + 6\lceil \frac{T(n)}{r} \rceil &\leq n + \frac{n}{r} + 6\frac{T(n)}{r} + 7 \\ &= n[1 + \frac{1}{r}] + 7 + 6\frac{T(n)}{r} \\ &\leq 2n + 7 + 6\frac{T(n)}{r} \\ &< 3n + 6\frac{T(n)}{r} \end{aligned}$$

Esto sucede pues $r \in \mathbf{N}$ y para $n > 7$.

Como

$$\inf_{n \to \infty} \frac{T(n)}{n} = \infty$$

entonces $\forall d \in \mathbf{R}\, \exists n_d \in \mathbf{N}$ tal que $\forall n > n_d\ \frac{T(n)}{n} > d$. Si tomamos $d = \frac{6}{c}$ entonces para $n > n_d$ sucede que:

$$\begin{aligned} 6\frac{T(n)}{r} + 3n &< 6\frac{T(n)}{r} + 3\frac{cT(n)}{6} \\ &< 6\frac{cT(n)}{12} + \frac{cT(n)}{2} \\ &= \frac{cT(n)}{2} + \frac{cT(n)}{2} \\ &= cT(n) \end{aligned}$$

pues escogimos $r \in \mathbf{N}$ tal que $rc > 12$ y por lo tanto el número de movimientos no excede $cT(n)$ para $n > n_d$.

Para completar la demostración es suficiente modificar M_2 de tal manera que verifique directamente todas las palabras de tamaño menor o igual a $max\{7, n_d\}$. ■

El Teorema 3 es el análogo al Teorema 1 (de compresión de cintas); pero estamos pidiendo condiciones adicionales, que no tomamos en cuenta para el primer teorema. La diferencia principal de trabajar con tiempo y espacio es intuitivamente que "el tiempo no se puede volver a usar y el espacio sí". Este hecho se refleja en el siguiente teorema, el cual es un buen ejemplo de la dificultad de trabajar con clases de complejidad definidas por el tiempo.

Teorema 4 *Sea $L = L_{M_1}$ para alguna MT M_1 con k cintas de complejidad tiempo acotado por $T(n) \Rightarrow$ L es aceptado por una MT M_2 con dos cintas en tiempo $T(n)\log T(n)$.*

Demostración :

Sea M_1 MT tal que $L_{M_1} \in$ **DTIEMPO(T(n))**. Construiremos una MT M_2 de 2 cintas que acepte a L en tiempo acotado por $T(n)\log T(n)$.

Cada símbolo del alfabeto de cinta de M_2 será una pareja de símbolos del alfabeto de M_1 unión $\{\alpha\}$ con α que no se encuentra en el alfabeto de M_1. El símbolo blanco de M_2 será el 2k-tuplo $(\alpha, b, \ldots, \alpha, b)$, donde b es el símbolo blanco de M_2. La primera cinta de M_2 tendrá dos pistas para cada cinta de trabajo de M_1; la segunda cinta se usará para marcar y transportar bloques de datos en la primera cinta. Por simplicidad, se centrará la atención en la simulación de una cinta particular de M_1, las demás cintas se simulan de la misma forma.

Una celda particular de la cinta 1 de M_2, llamada B_0, tendrá los símbolos examinados por cada una de las cabezas de M_1. Mas que mover las marcas para señalar las posiciones de las cabezas de M_1 (como en los teoremas anteriores), M_2 transportará los datos a través de B_0 en dirección contraria al movimiento de la cabeza lectora que se está simulando en M_1 (es como si "movieramos la cinta" en lugar de las cabezas lectoras). Así M_2 simulará cada movimiento de M_1 observando solamente a B_0. La cinta de M_2 estará dividida a la derecha de B_0 en $B_1, B_2, \ldots$ que incrementan su longitud en forma exponencial; esto es, la longitud del bloque B_i es 2^{i-1}. De la misma manera, a la izquierdad de B_0 estan los bloques $B_{-1}, B_{-2}, \ldots$ cuya longitud es 2^{-i-1}. Las divisiones de estos bloques no es física, esto es, no existe alguna marca dentro de las cintas que separe a los bloques; esta separación es más de construcción en la implemantación en la función de transición.

Sea a_0 que denote el contenido de la celda inicialmente examinada por la cinta en cuestión de M_1. Los contenidos de las celdas a la derecha serán $a_1, a_2, \ldots$ y a su izquierda $a_{-1}, a_{-2}, \ldots$. Los valores de las a_i's pueden cambiar cuando ellas entran a B_0. Como el símbolo blanco de M_2 es $(\alpha, b, \ldots, \alpha, b)$

entonces podemos suponer que las pistas superiores de M_2 están inicialmente vacías mientras que la pista inferior tiene a $\ldots, a_{-2}, a_{-1}, a_0, a_1, a_2, \ldots$ Aquí es pertinente notar que para nosotros el símbolo α representa ausencia de información. Que el símbolo b esté en alguna celda de M_2 no denota ausencia de información, sino que, ese símbolo b es el que se encuentra en la posición indicada por M_2 en M_1. Una celda en una pista de M_2 estará vacía si contiene a α.

Como se mencionó anteriormente, los datos en M_1 serán recorridos a través de B_0 y quizás modificados cuando pasen por él. Después de la simulación de cada movimiento de M_1 tendremos que mantener las siguientes condiciones:

1. Para todo $i > 0$ sucede necesariamente alguna de las tres condiciones siguientes:
   - B_i esta lleno (ambas partes) y B_{-i} vacío
   - B_i vacio y B_{-i} lleno
   - Las pistas inferiores de B_i, B_{-i} estan llenas mientras que las pistas superiores vacías
2. Los contenidos de B_i o B_{-i} representan celdas consecutivas en la cinta que en M_1 representan. $\forall i > 0$ la pista superior representa celdas a la izquierda de aquellas de la pista inferior. $\forall i < 0$ la pista superior representa celdas a la derecha de aquellas de la pista inferior.
3. Para $i < j$, B_i representa celdas a la izquierda de B_j.
4. B_0 siempre tiene la pista inferior llena y su pista superior especialmente marcada.

En este momento hay que notar que dada la manera en que definimos el alfabeto y símbolo blanco de M_2, al inicio de la simulación M_2 cumple con los puntos anteriores.

Para ver como se transfieren los datos, imaginemos que la cabeza de la cinta de M_1 (la que estamos observando) se mueve a la izquierda una celda, entonces M_2 debe correr los datos a la derecha. Para hacer esto, M_2 mueve la cabeza de su cinta 1 a B_0 y se mueve a la derecha hasta que encuentra el primer bloque B_i que no tiene ambas pistas llenas. Entonces M_2 copia todos los datos de $B_0, \ldots, B_{i-1}$ sobre la cinta 2 y almacena esta información en la pista inferior de B_i; esto si sucede que la pista inferior de B_i no esta

llena. Si la pista inferior de B_i esta llena, la pista superior de B_i se usa en lugar de la inferior. En ambos casos, existe el espacio justo para distribuir los datos según las condiciones antes mencionados, pues:

$$1 + 2\sum_{n=0}^{i-1} 2^n$$

es la cantidad de información en $B_0, \ldots, B_{i-1}$, que se mueve a $B_1, \ldots, B_i$ que es

$$\sum_{n=0}^{i} 2^n$$

y

$$1 + 2\sum_{n=0}^{i-1} 2^n = \sum_{n=0}^{i} 2^n$$

También hay que notar que los datos pueden ser tomados y almacenados en sus nuevas localidades en tiempo proporcional a la longitud de B_i (recorriendo cada bloque que se movió 3 veces).

A continuación, en tiempo proporcional a lo longitud de B_i, M_2 encuentra B_{-i} (usando la cinta 2 para medir la distancia de B_i a B_0). Si B_{-i} esta completamente lleno, M_2 toma la pista superior de B_{-i} y lo almacena en la cinta 2. Si B_{-i} esta lleno a la mitad, la pista inferior se coloca en la pista 2. En ambos casos, lo que se ha copiado en la cinta 2 se copia en las pistas inferiores de $B_{-i-1}, \ldots, B_0$ (por la regla dos, estas pistas estan vacías, puesto que $B_1, \ldots, B_{i-1}$ están llenas). Otra vez, existe el espacio justo para almacenar los datos. Todas las operciones de arriba se realizan en tiempo proporcional a la longitud de B_i.

La operaciones anteriormente descritas distribuyen los datos de manera que los puntos 1), 2) y 3) se satisfacen y mantienen. Llamaremos las operaciones anteriores en su conjunto como una B_i – *operación*. El caso en que la cabeza de M_1 se mueve a la derecha es simétrico.

Notamos que para cada cinta de M_1, M_2 realiza una B_i – *operación* a lo más una vez por cada 2^{i-1} movimientos de M_1, puesto que esta es la longitud de $B_1, \ldots, B_{i-1}$, los cuales están llenos en su parte inferior después de una B_i – *operación* y deben llenarse para realizar el siguiente movimiento. También una B_i – *operación* no puede realizarse antes del primer 2^{i-1} movimiento de M_1. Así, si M_1 opera en tiempo $T(n)$, M_2 realizará a lo más B_i – *operaciones* para aquellas i tales que $i \leq \log_2 T(n) + 1$ (es decir $2^{i-1} \leq T(n)$).

Con lo anterior hemos visto que existe una constante m tal que M_2 hace $m2^i$ movimientos a lo más al realizar una B_i – *operación*. Si M_1 hace $T(n)$ movimientos, M_2 hace a lo más :

$$T_2(n) = \sum_{i=1}^{\lceil \log_2 T(n)+1 \rceil} m2^i \frac{T(n)}{2^{i-1}}$$

para simular una cinta de M_1. Entonces:

$\Longrightarrow$

$$\begin{aligned} T_2(n) &= \sum_{i=1}^{\lceil \log_2 T(n)+1 \rceil} m2T(n) \\ &= 2mT(n)\lceil \log_2 T(n) + 1 \rceil \\ &< 2mT(n)\{\log_2 T(n) + 2\} \\ &= 2mT(n)\log_2 T(n) + 4mT(n) \\ &< 4m\{T(n)\log_2 T(n) + T(n)\} \\ &\leq 4mT(n)\log_2 T(n) \end{aligned}$$

que sucede si $\log_2 T(n) \geq 1$, es decir, si $T(n) \geq 1$. Esto quiere decir que debemos implementar en la función de transición de M_2 directamente si $\epsilon \in L$.

Entonces por el teorema 3 podemos modificar M_2 para que corra en tiempo no mayor a $T(n)\log T(n)$. ■

3.3 Relaciones entre Clases de Complejidad

En esta sección presentaremos algunas relaciones que existen entre las clases de complejidad que hemos estudiado hasta ahora y presentaremos el Teorema de Savitch, el cual nos enuncia una relación sorprendente entre las clases de complejidad de espacio no determinista y espacio determinista.

Dado que la Máquina de Turing es nuestro dispositivo de cómputo, entonces todo lo que es calculable en nuestro dispositivo adquiere gran importancia para nosotros. En este contexto ahora definimos lo que para nosotros será una función *bien comportada* (en nuestro sentido).

Definición 11 (Espacio Constructividad.) *Sea* $S : \mathrm{N} \longrightarrow \mathrm{N}$. *Decimos que* S *es* espacio construible *si existe una Máquina de Turing* M *que está acotada en espacio por* $S(n)$ *en cualquiera de sus cintas, cuando se le dá como entrada cualquier palabra de longitud* n.

Definición 12 (Espacio Constructividad Total.) *Sea* $S : \mathrm{N} \longrightarrow \mathrm{N}$. *Decimos que* S *es* espacio construible total *si existe una Máquina de Turing* M *que utiliza exactamente* $S(n)$ *celdas sobre cualquiera de sus cintas, cuando se le dá como entrada cualquier palabra de longitud* n.

Presentemos el siguiente lema que nos afirma que todo lenguaje que se encuentra en alguna clase de complejidad de espacio es recursivo.

Lema 2
Sea L *un lenguaje recursivo tal que* L *tiene Complejidad de Espacio* $S(n)$, *con* $S(n) \geq \log_2 n$ *y* $S(n)$ *espacio construible, entonces existe una Máquina de Turing* M, *tal que* M *acepta a* L *y* M *se detiene para cualquier palabra de entrada.*

Demostración :
Sea M_1 MT tal que $L = L_{M_1} \in \mathbf{DESPACIO}(S(n))$ con s estados y t símbolos de cinta. El máximo número de pasos en los que M_1 acepta una palabra de L son $(n+2)sS(n)t^{S(n)}$; esto es, $n+2$ posiciones en la cinta de entrada, s estados, espacio de trabajo $S(n)$ y $t^{S(n)}$ posibles contenidos de la cinta de trabajo (podemos pensar en una MT con una sola cinta de trabajo, en virtud del teorema de compresión de cintas). Afirmamos que el número de movimientos que realiza M_1 antes de aceptar a $\omega \in L$, son $(n+2)sS(n)t^{S(n)}$, pues es el número total de posibles descripciones instantáneas de M_1 y en caso de que M_1 ejecute más de estos pasos, entonces se estaría repitiendo alguna descripción instantánea, lo cual indicaría que la máquina ha caído en un ciclo infinito, puesto que estamos trabajando en una clase determinista.

Construimos M_2 con una cinta adicional a la de M_1, en la cual simularemos un contador que parará después de $(4st)^{S(n)} \geq (n+2)sS(n)t^{S(n)}$ movimientos. El contador tendrá longitud $\lceil \log_2 n \rceil$ y cuenta en base $4st$. Si M_1 examina una nueva celda más allá de las celdas que contienen al contador entonces se incrementa la longitud del contador. Así, si M ha usado i celdas de cinta entonces el contador lo detectará cuando el contador alcanze $(4st)^{max\{i,\log_2 n\}}$ el cual es a lo más $(n+2)sS(n)s^{S(n)}$. ■

Con este lema como herramienta presentamos los siguientes resultados.

Proposición 1
a) Sea $T(n)$ recursiva, entonces

$$\mathbf{DTIEMPO(T(n))} \subseteq \mathbf{NTIEMPO(T(n))}$$

y

$$\mathbf{DESPACIO(T(n))} \subseteq \mathbf{NESPACIO(T(n))}$$

b) Sea $T(n)$ recursiva, entonces

$$\mathbf{DTIEMPO(T(n))} \subseteq \mathbf{DESPACIO(T(n))}$$

y

$$\mathbf{NTIEMPO(T(n))} \subseteq \mathbf{NESPACIO(T(n))}$$

c) Si $S'(n) \in O(S(n))$ entonces

$$\mathbf{DESPACIO(S'(n))} \subseteq \mathbf{DESPACIO(S(n))}$$

y

$$\mathbf{NESPACIO(S'(n))} \subseteq \mathbf{NESPACIO(S(n))}$$

d) Si $T'(n) \in O(T(n))$ con $n \in O(T(n))$ entonces

$$\mathbf{DTIEMPO(T'(n))} \subseteq \mathbf{DTIEMPO(T(n))}$$

y

$$\mathbf{NTIEMPO(T'(n))} \subseteq \mathbf{NTIEMPO(T(n))}$$

e) Si $S(n)$ es espacio construible y $S(n) \geq \log n$ entonces

$$\exists\, c \textit{ tal que } \mathbf{DESPACIO(S(n))} \subseteq \mathbf{DTIEMPO(c^{S(n)})}$$

f) Si $T(n)$ es tiempo construible entonces

$$\exists\, c \textit{ talque } \mathbf{NTIEMPO(T(n))} \subseteq \mathbf{DTIEMPO(c^{T(n)})}$$

Demostración :

a) Son inmediatas ambas afirmaciones, pues toda máquina determinista puede siempre ser considerada como no determinista.

b) Sea $L \in \mathbf{DTIEMPO(T(n))}$ entonces $\exists M$ MT tal que $L = L_M$. Como $L_M \in \mathbf{DTIEMPO(T(n))}$ entonces M examina a lo más $T(n) + 1$ celdas sobre cualquiera de sus cintas. En virtud del Teorema 1 podemos construir una MT M_1 que ocupe $\lceil \frac{T(n)+1}{2} \rceil$ celdas (modificando el alfabeto para que guarde dos símbolos por celda). Así el espacio requerido es $\lceil \frac{T(n)+1}{2} \rceil$ celdas, que es a lo más $T(n)$. El mismo razonamiento se utiliza para el caso no determinista.

c) Por definición, si $S'(n) \in O(S(n))$ entonces $\exists c > 0$ tal que $\forall n > N$ $S'(n) \geq cS(n)$; de aqui que cualquier lenguaje que pertenezca a la clase $\mathbf{DESPACIO(S'(n))}$ también se encuentre en $\mathbf{DESPACIO(cS(n))}$. Por el Teorema de compresión de cinta podemos afirmar que si L se encuentra en $\mathbf{DESPACIO(S'(n))}$ entonces

$$L \in \mathbf{DESPACIO(cS(n))} \frac{1}{c}) \forall c.$$

El mismo argumento se utiliza en el caso no determinista.

d) Si utilizamos el teorema de aceleración lineal con los mismos, razonamientos de c) y utilizando que $n \in O(T(n))$ llegamos al resultado requerido.

e) Sea $L \in \mathbf{DESPACIO(S(n))}$ entoces $L = L_M$ para alguna MT M tal que $L_M \in \mathbf{DESPACIO(S(n))}$. Si M tiene k estados, t síbolos de cinta y usa $S(n)$ espacio, entonces el número de descripciones instantaneas de M sobre una entrada de longitud n es a lo más $k(n+2)S(n)t^{S(n)}$. Puesto que $S(n) \geq \log_2(n)$ entonces $\exists\, c$ tal que $\forall\, n \geq 1 \quad c^{S(n)} \geq k(n+2)S(n)t^{S(n)}$ (lema 1).

Construyamos de M una MT M_1 que use una cinta para contar $c^{S(n)}$ y otras dos para simular a M. Si M_1 no ha aceptado cuando el contador ha alcanzado $c^{S(n)}$ entonces para sin aceptar, despues de este momento, M_1 debería de haber repetido alguna descripción instantanea y haber caído en un ciclo por lo cual nunca aceptará. Claramente, por el contador, M_1 es de tiempo acotado $c^{S(n)}$.

f) Sea M una MT no determinista de tiempo acotado $T(n)$ con s estados, t símbolos de cinta y k cintas. El número de posibles descripciones instantaneas de M, dada una entrada ω de longitud n, son al lo más

$$s(T(n)+1)^k t^{kT(n)}$$

que es el producto del número de estados, posiciones de la cabeza y contenido en las k cintas. Si tomamos $d = s(t+1)^{3k}$ entonces:

$$d^{T(n)} \geq s(T(n)+1)^k t^{kT(n)} \qquad \forall\, n \geq 1$$

Una MT multicintas determinista puede determinar si M acepta una entrada ω con $\mid \omega \mid = n$, construyendo una lista de todas las descripciones instantaneas de M que son alcanzables desde la descripción inicial. Este proceso puede llevarse a cabo en tiempo acotado por el cuadrado de la longitud de la lista. Puesto que la longitud de la lista de descripciones instantaneas de a lo más $d^{T(n)}$ símbolos, entonces el tiempo esta acotado por $c^{T(n)}$ para alguna constante c. ■

Teorema 5 *[Teorema de Savitch]. Sea $S(n)$ una función espacio construible total tal que $S(n) \geq \log_2 n$ entonces*

$$\mathbf{NESPACIO(S(n)) \subseteq DESPACIO(S^2(n))}$$

Demostración :

Como estamos trabajando con espacio consideremos una MT con dos cintas; una de entrada y otra de trabajo. Sea $L = L_{M_1}$ tal que $L_{M_1} \in$ **NESPACIO(S(n))**. El número de descripciones instantaneas para una entrada de longitud n es a lo más $c^{S(n)}$, para alguna constante c. Así, si M_1 acepta, lo hará en una sucesión de a lo más $c^{S(n)}$ pasos.

Denotemos mediante $I_1 \overset{(i)}{\longmapsto} I_2$ el hecho de que de la descripción instantanea I_1 se pueda llegar en a lo más 2^i movimientos a la descripcion instantanea I_2. Ahora, para $i \geq 1$ podemos determinar si $I_1 \overset{(i)}{\longmapsto} I_2$ probando para cada I' si $I_1 \overset{(i-1)}{\longmapsto} I'$ y $I' \overset{(i-1)}{\longmapsto} I_2$. De aquí que el espacio necesario para determinar si $I_1 \overset{(i)}{\longmapsto} I_2$ es el mismo que el que se necesita para grabar la descripción instantanea I', que esta siendo probada, más el espacio necesario para probar si se puede obtener cualquier otra en 2^{i-1} movimientos. Notemos el hecho de que el espacio necesario para probar si alguna descripción instantanea es alcanzable desde otra en 2^{i-1} movimientos se puede reutilizar.

El siguiente algoritmo, basado en las ideas antes expuestas, determina si la palabra $\omega \in L_{M_1}$:

begin

Sea $\mid \omega \mid = n$

Sea I_0 ID de M_1 con ω
for each ID final I_f de longitud a lo más $S(n)$ **do**
if TEST$(I_0, I_f, \lceil \log_2 c \rceil)$ **then acepta**

end

procedure TEST(I_1, I_2, i)

if $i = 0$ **and** $(I_1 = I_2)$ **or** $I_1 \overset{(0)}{\longmapsto} I_2$ **then**
return true
if $I \geq 1$ **then**
for each ID I' de longitud a lo más $S(n)$ **do**
if TEST$(I_1, I', i-1)$ **and** TEST$(I', I_2, i-1)$ **then**
return true

return false

end TEST

Se puede observar que la idea básica de la simulación está en el procedimiento TEST(I_1,I_2,i). Este procedimiento decide si existe alguna forma de acceder de I_1 a I_2 en a lo más 2^i pasos; esto lo hace probando si existe alguna configuración I' tal que se pueda acceder de I_1 a I' en 2^{i-1} pasos y de I' a I_2 en 2^{i-1} pasos, es decir en total 2^i pasos.

El algoritmo completo puede implementarse en una MT M_2 que use una pila de registros que contengan la información de los parámetros del procedimiento TEST. Para cada llamada, los registros de esta pila contendrán a I_1, I_2 e i; así como los valores de la variable local I'. Como I_1, I_2 e I' son descripciones instantaneas que no tienen más de $S(n)$ celdas entonces podemos representarlas (a las tres simultaneamente) en $S(n)$ celdas, esto para la cinta de trabajo. Para la cinta de entrada, basta con indicar la posición de su cabeza lectora; pues dado el modelo que estamos trabajando la cinta es de sólo lectura y entonces si en M_2 se tiene la misma entrada que en M_1 se puede conocer el símbolo que se lee sobre esta cinta. El parámetro i puede codificarse en $\lceil \log_2 c \rceil S(n)$ celdas. Así cada registro toma espacio en $O(S(n))$.

Como i se decrementa para cada llamada del procedimiento TEST, la llamada inicial es con $i = \lceil \log_2 c \rceil S(n)$ y se realiza hasta que $i = 0$ entonces

el número máximo de registros en la pila es de $O(S(n))$. Así el espacio total usado es $O(S^2(n))$. Por le teorema 1 podemos reducir M_2 para que acepte en exactamente $S^2(n)$ espacio. ■

Presentemos la siguiente proposición que nos proporciona algunas relaciones entre las clases de complejidad presentadas en 3.3.1.

Proposición 2 *a) Cada clase determinista es cerrada bajo complemento.*
b) Cada clase determinista esta incluida en su correspondiente no determinista.
c) **PESPACIO = NPESPACIO.**

Demostración :

a) Si cambiamos los estados de aceptación por estados finales de rechazo entonces aceptamos el complemento del lenguaje y no cambiamos los recursos con los que se esta trabajando.

b) Por la proposión 1a).

c) Por el teorema de Savitch $\forall i$ $\mathbf{NESPACIO(n^i)} \subseteq \mathbf{DESPACIO(n^{2i})}$. Así :

$$\bigcup_{i\geq 0} \mathbf{NESPACIO(n^i)} \subseteq \bigcup_{i\geq 0} \mathbf{DESPACIO(n^{2i})} = \bigcup_{c\geq 0} \mathbf{DESPACIO(n^c)}$$

y como

$$\mathbf{DESPACIO}(n^i) \subseteq \mathbf{NPESPACIO}(n^i)\ \forall\, i \geq 0$$

entonces

$$\bigcup_{i\geq 0} \mathbf{DESPACIO(n^i)} \subseteq \bigcup_{i\geq 0} \mathbf{NESPACIO(n^i)}$$

y por lo tanto

$$\mathbf{DESPACIO = NPESPACIO.}$$

■

El inciso c) completa el Teorema de Savitch, en el sentido que demuestra la igualdad **PESPACIO = NPESPACIO.** Este es un hecho muy interesante pues nos presenta una relación entre clases deterministas y no deterministas afirmando que la clase de lenguajes los cuales son computables en

"espacio polinomial no determinista" es igual a la clase de lenguajes computables en "espacio polinomial determinista".

En Teoría de Computación uno de los problemas más importantes que existen es determinar si $\mathbf{P} = \mathbf{NP}$. Una de las contenciones es fácil: $\mathbf{P} \subseteq \mathbf{NP}$ puesto que toda Máquina de Turing Determinista puede tomarse como No Determinsta. La otra contensión $\mathbf{NP} \subseteq \mathbf{P}$ es la que resulta realmente interesante, pués puede interpretarse como sigue: Dado que conocemos la forma de verificar en tiempo polinomial si una entrada ω es solución de un problema P (es decir, $P \in \mathbf{NP}$) entonces ¿Se puede encontrar una Máquina de Turing Determinista (algoritmo determinista) tal que pueda encontrar alguna solución del problema dado?. El hecho de que $\mathbf{IP} = \mathbf{PSPACE}$ tiene, como se verá más adelante, grandes implicaciones en la respuesta de esta pregunta.

Capítulo 4

PESPACIO completez e IP

En el presente capítulo hablaremos de.REDUCIBILIDAD POLINOMIAL como una forma de comparar la dificultad de reconocimiento en distintos lenguajes. Este idea también nos permitirá formalizar el concepto de los elementos "más difíciles" dentro de las clases de complejidad. Estudiaremos en particular el problema de las Fórmulas Booleanas Cuantificadas (*QBF*) y veremos que en **PESPACIO**, este lenguaje es de los "más difíciles". También nos introduciremos en los Sistemas Interactivos de Prueba, que son modelos de cómputo a partir de los cuales definiremos la clase **IP** de lenguajes. Las dos primeras secciones y la última se realizaron en base a la exposición que se hace de los temas presentados en [10] y [12]. La tercera sección contiene una prueba de que *QBF* es **PESPACIO**-completo, esta prueba descansa, principalmente, en la Proposición 4, la cual es una modificación de los resultados expuestos en [10] y [12]. La proposición es un resúmen y clarificación de las ideas propuestas en [10] y [12] (esta proposición como tal no se encuentra en la bibliografía citada). El ejemplo del Sistema Interactivo de Prueba para el problema de Gráficas No Isomorfas fué tomado de [14].

4.1 Transformaciones Polinomiales

Definición 13 (Transformación Polinomial) *Sea L_1 y L_2 dos lenguajes sobre los alfabetos Σ_1 y Σ_2, respectivamente. Una* Transformación Polinomial *de L_1 a L_2 es una función $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ que satisface :*

1. *$\exists M$ Máquina de Turing Determinista que trabaja en tiempo polinomial que calcula a f.*

2. $\forall \omega \in \Sigma_1^*, \omega \in L_1 \iff f(\omega) \in L_2$

Si existe una transformación polinomial de L_1 a L_2 lo denotaremos $L_1 \preceq_f L_2$. Hay que notar el hecho de que existen otros tipos de transformaciones que no son del tipo polinomial (por ejemplo logarítmico). A las Tranformaciones entre lenguajes también se les llama *reducciones.*

Al estar hablando de reducibilidad tenemos que pensar que estas reducciones se realizan con respecto a una codificación específica, puesto que el esquema de codificación puede afectar fuertemente el hecho de transformar un lenguaje a otro. Por esto, las reducciones en realidad se realizan de $L(e_1) \preceq_f L(e_2)$, donde e_1 y e_2 son los esquemas de codificación de L_1 y L_2, respectivamente. Demostraremos ahora las siguientes propiedades acerca de este tipo de reducibilidad.

Lema 3 *Sea* $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, $L_3 \subseteq \Sigma_3^*$ *con* Σ_1, Σ_2, Σ_3 *alfabetos finitos, entonces:*

a) $\preceq_f$ *es reflexiva y transitiva.*
b) $L_1 \preceq_f L_2 \iff L_1^c \preceq_f L_2^c \qquad \forall L_1, L_2$

Demostración:

a) Sea $L \subseteq \Sigma^*$ entonces $L \preceq_f L$, si tomamos a $f(\omega) = \omega \quad \forall \omega \in \Sigma^*$. Por lo tanto $\preceq_f$ es reflexiva y f es de tiempo polinomial.

Sea $L_1 \preceq_{f_1} L_2$, $L_2 \preceq_{f_2} L_3$ entonces definamos a $f : \Sigma_1^* \longrightarrow \Sigma_3^*$ como $f(\omega) = f_2(f_1(\omega)) \quad \forall \omega \in \Sigma_1^*$. Claramente $f(\omega) \in L_3$ si y solo si $\omega \in L_1$. Ahora, como f_1 se puede calcular en tiempo polinomial entonces $\exists M_1$ Máquina de Turing Determinista tal que M_1 calcula a f_1 en tiempo polinomial, como en cada movimiento de M_1 a lo más puede imprimir un símbolo, entonces $\mid f_1(\omega) \mid \leq p_1(\mid \omega \mid)$, donde $p_1(n)$ es un polinomio que acota el número de movimintos de M_1 para todas las entradas de longitud n. Como $f_1(\omega) \in L_2$ entonces ahora le aplicamos a f_2. Para $f_2 \exists M_2$ Máquina de Turing Determinista tal que M_2 calcula a f_2 en tiempo polinomial. Así, si aplicamos f_2 a $f_1(\omega)$ entonces $\mid f_2(f_1(\omega)) \mid \leq p_2(p_1(\mid \omega \mid))$, donde $p_2(n)$ es un polinomio que acota los movimientos de M_2 para las entradas de longitud n. De aquí que $f = f_2 \circ f_1$ se puede calcular en $O(p_1(\mid \omega \mid) + p_2(p_1(\omega)))$. Por lo tanto tp es transitiva.

b) Sea $L_1 \preceq_f L_2$, como $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ entonces $\forall \omega \in L_1^c \iff L_2^c$, esto es cierto por ser f reducción polinomial y f sigue siendo calculable polinomialmente. ■

La relación $\preceq_f$ define un orden parcial sobre las Clases de Complejidad y podemos entonces definir clases de equivalencia de la siguiente manera: $L_1 \equiv L_2$ si y sólo si $L_1 \preceq_f L_2$ y $L_2 \preceq_g L_1$. De esta manera tenemos particiones dentro de nuestras clases de lenguajes. Con estas ideas en mente podemos considerar, intuitivamente, que un lenguaje L es difícil de reconocer dentro de una clase de complejidad C, si todo lenguaje en C es reducible mediante una reducción fácilmente computable a L (si tenemos definida una clase que utiliza cierta cantidad de recursos de algún tipo, entonces una "reducción fácil" se puede conceptualizar como aquella que no utiliza más poder de los recursos en cuestión). Ahora bién, si aceptamos esta idea entonces debemos notar que esta noción de "lenguajes difíciles" está dada con respecto a una reducción particular.

4.2 Clases Completas y Difíciles

Definición 14 *Sea C una clase de lenguajes, entonces definimos:*

$$a)\mathcal{C}\text{-completo} = \{L \in C \mid \forall L' \in C \exists \preceq_f \textit{ tal que } L' \preceq_f L\}$$
$$b)\mathcal{C}\text{-difícil} = \{L \mid \forall L' \in C \exists \preceq_f \textit{ tal que } L' \preceq_f L\}$$

Nuevamente, estas definiciones están en función de una reducción específica (en este caso de tiempo polinomial). De la definición de *C-difícil* hay que notar que no es necesario que los lenguajes de esta clase pertenezcan a C; mientras que para C-completo sí es necesario. En lo que resta, cuando aparezaca $\preceq_f$ nos referiremos explícitamete a una reducción de tipo polinomial.

La utilidad de la siguiente definición se verá con la siguiente proposición .

Proposición 3 *Sea C cualquier clase de complejidad, entonces :*

1. *Si $L_1 \in C$-difícil y $L_1 \preceq_f L_2$ entonces $L_2 \in C$-difícil.*
2. *Si $L_1 \in C -$ completo y $L_1 \preceq_f L_2 \Rightarrow L_2 \in C -$ completo.*
3. **P** *y* **PESPACIO** *son cerrados bajo reducciones polinomiales.*

Demostración :

1. Como $L_1 \in C$-*difícil* $\Rightarrow \forall L \;\; L \preceq_f L_1$ y como $L_1 \preceq_f L_2$ entonces por transitividad de $\preceq_f \forall L \, L \preceq_f L_2$. Por lo tanto, $L_2 \in C$-*difícil*

2. El mismo argumento que en 1.

3. Lo que queremos demostrar es que si $L_1 \preceq_f L_2$ y $L_2 \in$ **PESPACIO** entonces $L_1 \in$ **PESPACIO**, es decir, $\exists M_1$ Máquina de Turing Determinista tal que $L_1 = L_{M_1} \in$ **PESPACIO**.

Que $L_1 \preceq_f L_2 \Rightarrow \exists f : \Sigma_1^* \longrightarrow \Sigma_2^*$ tal que :

- $\exists M_f$ Máquina de Turing Determinista de tiempo polinomial que la calcula.
- $\forall \omega \in \Sigma_1^*, \omega \in L_1 \Leftrightarrow f(\omega) \in L_2$

Que $L_2 \in$ **PESPACIO** implica que

- $\exists M_2$ Máquina de Turing Determinista tal que

$$L_2 = L_{M_2} \in \textbf{PESPACIO}.$$

Definamos M_1 de la siguiente manera:

Sea $\omega \in \Sigma_1^*$, apliquemos M_f a ω, entonces obtendremos $f(\omega) \in L_2$; el cómputo de de M_f sobre ω está acotado por un polimonio p y, de aquí que, $| f(\omega) | \leq p(| \omega |)$. Sobre $f(\omega)$ aplicamos M_2, como $M_2 \in$ **PESPACIO** entonces el espacio necesario para el cómputo de M_2 sobre $f(\omega)$ está acotado por algún polinomio q; es decir, el espacio que utiliza M_1 es $O(q(p(| \omega |)))$.

Se tiene que $L_{M_1} = L_1$, pues sea $\omega \in L_1$, M_1 acepta a ω porque M_2 acepta a $f(\omega) \in L_2$ y $\omega \in L_1 \Leftrightarrow f(\omega) \in L_2$. El razonamiento para el tiempo es análogo. ■

La anterior proposición nos lleva entonces a concluir que para encontrar el conjunto de los problemas más difíciles dentro de una clase de complejidad determinada, podría comenzar encontrando uno de sus elementos y posteriormente para verificar que un elemento esté allí realizar una reducción a algún lenguaje que sepamos está dentro de la clase completa.

4.3 PESPACIO-completo

En esta sección demostraremos que el problema de las *Fórmulas Booleanas Cuantificadas QBF* es **PESPACIO**-completo. Comenzaremos por definirlo.

Definición 15 *Una Variable Booleana es cualquier símbolo al cual puede asociarse alguno de los valores 0 ó 1.*

Al 0 se le asocia el valor *falso* y a 1 el valor de *verdadero*.

Definición 16 *Sea X un conjunto numerable de variables booleanas. La clase de las "Fórmulas Booleanas" sobre X es la clase más pequeña definida por:*

1. *Las constantes 0 y 1 son constantes booleanas.*
2. *$\forall x \in X$, x es fórmula booleana.*
3. *Si F_1 y F_2 son fórmulas booleanas entonces $F_1 \wedge F_2$, $F_1 \vee F_2$ y $\neg F_1$ son fórmulas booleanas.*

   *El conjunto de las "Fórmulas Booleanas Cuantificadas" (*QBF *) sobre X se define por las siguientes reglas adicionales:*
4. *Toda fórmula booleana es fórmula booleana cuantificada.*
5. *Si $x \in X$ y F es fórmula booleana cuantificada entonces $\exists x\, F$ y $\forall x\, F$ son fórmulas booleanas cuantificadas.*

Dada una *QBF* una variable que ocurre en la fórmula pero que no está afectada por algún cuantificador se le llama *variable libre.*

Definición 17 *Una* asignación booleana *es un mapeo de X al $\{0,1\}$. Dada una fórmula booleana cuantificada F y una asignación V, el valor de F bajo V es el valor booleano definido como sigue:*

1. $V(F) = 0$ *si* $F = 0$
2. $V(F) = 1$ *si* $F = 1$
3. $V(F) = V(x)$, $F = x$ *para* $x \in X$
4. $V(F) = V(F_1) \vee V(F_2)$ *si* $F = (F_1 \vee F_2)$
5. $V(F) = V(F_1) \wedge V(F_2)$ *si* $F = (F_1 \wedge F_2)$
6. $V(F) = \neg V(F_1)$ *si* $F = \neg F_1$
7. $V(F) = V(F_1 \mid_{x:=0}) \vee V(F_1 \mid_{x:=1})$ *si* $F = \exists x\, F_1$
8. $V(F) = V(F_1 \mid_{x:=0}) \wedge V(F_1 \mid_{x:=1})$ *si* $F = \forall x\, F_1$

Donde $F \mid_{x:=a}$ es la fórmula booleana que se obtiene sustituyendo a a por toda ocurrencia libre de x en F.

Definición 18 *Dada un fórmula booleana F, decimos que una asignación V* satisface *si V(F) = 1. Una fórmula es* satisfacible *si existe una asignación V la cual satisfaga a F; de otra manera F no se puede satisfacer.*

En estos momentos ya estamos listos para definir formalmente el lenguaje *QBF*.

Definición 19 *Sea* Σ *un alfabeto fijo tal que las fórmulas booleanas cuantificadas se pueden codificar sobre él, entonces* QBF *es el lenguaje de todas las codificaciones de fórmulas booleanas cuantificadas, sin variables libres, las cuales se pueden evaluar a verdadero.*

Dado un alfabeto y una codificación fijos, la longitud de una fórmula es la longitud de la palabra en el alfabeto y la codificación dada. Es claro que el número de ocurrencias de una variable en una fórmula es de orden lineal en el número de conectivos; y si una fórmula contiene n variables entonces éstas se pueden codificar en $\lceil \log n \rceil$ símbolos. Así, si una fórmula contiene m conectivos entonces su longitud es de orden $m \log m$.

Ahora que hemos definido a *QBF*, procedamos a demostrar que este lenguaje es **PESPACIO**-completo.

Lema 4 QBF $\in$ **PESPACIO**

Demostración :

El siguiente algoritmo recursivo describe una manera de evaluar una fórmula booleana cuantificada.

evaluación(F)

Si F **es**

- *VERDADERO*: **return** *VERDADERO*
- *FALSO*: **return** *FALSO*
- $\neg F'$: **return** $\neg$**evaluación**(F')
- F' *op* F'' con *op* operador booleano
  - **evaluación**(F')
  - **evaluación**(F'')
  - **return evaluación**(F') *op* **evaluación**(F'')
- $\forall x F'$:

- **evaluación**($F'|_{x:=0}$)
- **evaluación**($F'|_{x:=1}$)
- **return** *AND* de ambos resultados

• $\exists x F'$:

- **evaluación**($F'|_{x:=0}$)
- **evaluación**($F'|_{x:=1}$)
- **return** *OR* de ambos resultados

end evaluación

main program
z := **evaluación**(F)
si z = *TRUE* **entonces** *acepta* **sino** *rechaza*
end program

Si $|F| = n$ entonces el número de variables más el número de operadores es a lo más n. Así la profundidad de la recursión es a lo más n (puesto que F no tiene variables libres). Si usamos la misma técnica del Teorema de Savitch (registros de activación) entonces este procedimiento puede implementarse en espacio $O(n^2)$. De aquí que $QBF \in$ **PESPACIO.** ■

Dada una descripción instantánea $X_1 \cdots \overset{q}{X_k} \cdots X_m$ de una *MT* M, para describirla en términos de fórmulas booleanas podemos definir las variables C_{iX}, donde $1 \leq i \leq m$ y X es un símbolo de cinta ó un símbolo compuesto por un estado de M y un símbolo de cinta. Estas variables serán verdaderas si y sólo si en el i-esimo lugar de la descripción instantánea aparece el símbolo X. Ahora bien, si tenemos m de estas variables entonces cada una de ellas podemos codificarlas en $1 + \lceil \log m \rceil$ espacio y por lo tanto a todas ellas en $m(1 + \lceil \log m \rceil)$ espacio. Estas variables las utilizaremos en la siguiente proposición que nos muestra cómo expresar ciertas propiedades de los cómputos de una *MT* por medio de fórmulas booleanas.

Proposición 4 *Sea M una MT con una sola cinta que trabaja en espacio polinomial. Sea p un polinomio que permite que cualquier configuración de M, con entradas de longitud n, sea de longitud $p(n)$. Definimos C y D conjuntos de variables booleanas C_{iX}, D_{iY} como se definieron anteriormente; y sean $\alpha = \alpha_1 \cdots \alpha_{p(n)}$ y $\beta = \beta_1 \cdots \beta_{p(n)}$ con $\alpha_i \in C$ y $\beta_i \in D$ con α_i = verdadero $= \beta_i$. Entonces podemos construir en tiempo polinomial las siguientes fórmulas booleanas.*

1. *$Config(\alpha)$ = true ssi α codifica una configuración de M.*
2. *$Next(\alpha,\beta)$ = true ssi α y β codifican configuraciones de M y se puede llegar de α a β en un movimiento válido de M.*
3. *$Equal(\alpha,\beta)$ = true ssi α y β codifican configuraciones válidas de M.*
4. *$Initial(\alpha,\omega)$ = true ssi α codifica la configuración inicial de M con entrada ω.*
5. *$Accepts(\alpha)$ = true ssi α codifica alguna configuración final de M.*

Demostración :

1) Para que las C_{iX} codifiquen una configuración de M debe de suceder que para cada i exactamente una de las C_{iX} sea verdadera. Así dentro de $\alpha = \alpha_1, \cdots, \alpha_{p(n)}$ debe suceder que cada α_i sea alguna de C_{iX} y que $\alpha_i \neq \alpha_j$, $i \neq j$.

$$F = \left(\bigwedge_{i=1}^{p(n)} \left[\bigvee_{X} C_{iX} \wedge \neg \left(\bigvee_{X \neq Y} (C_{iX} \wedge C_{iY}) \right) \right] \right) \wedge \left(\neg \left(\bigwedge_{\substack{i \neq j \\ X,Y \in C_Q}} (C_{iX} \wedge C_{iY}) \right) \wedge \bigwedge_{\substack{i \\ X \in C_Q}} C_{iX} \right)$$

Para que esta fórmula se satisfaga tiene que suceder que para cada $1 \leq i \leq p(n)$ se satisfaga

$$\bigvee_{X} C_{iX} \wedge \neg \left(\bigvee_{X \neq Y} (C_{iX} \wedge C_{iY}) \right)$$

esto es, para cada i exactamente una X debe ser verdadera, es decir, que en cada lugar $1 \leq i \leq p(n)$ sólo exista un símbolo. Además se debe de cumplir que exista un sólo símbolo compuesto (estado). Para esto se debe de satisfacer la segunda parte de la fórmula:

$$\left(\neg \left(\bigvee_{\substack{i \neq j \\ X,Y \in C_Q}} (C_{iX} \wedge C_{jY}) \right) \wedge \bigvee_{\substack{i \\ X \in C_Q}} C_{iX} \right)$$

donde C_Q es el conjunto de símbolos compuesto por un estado y un símbolo. Esta fórmula obliga a que exactamente un estado aparezca en la configuración que queremos representar.

Esta fórmula se satisface si se dá una configuración de M y se mapea a las C_{iX}, y si un conjunto de C_{iX} satisfacen a F entonces representan una configuración de M.

Como la longitud de F es de $O(p^2(n))$ entonces F puede construirse en tiempo polinomial.

2) Esta fórmula debe de comenzar con $Config(\alpha) \wedge Config(\beta)$. Si ya se verificó que α y β codifiquan configuraciones válidas de M, entonces que β sea la siguiente configuración de α depende de la función de trancisión δ de M y de los símbolos adyacentes al símbolo compuesto de α. Además α y β son iguales excepto en el movimiento que define la función de trancisión δ. Habiendo dicho lo anterior podemos especificar para cada función de trancisión y alfabeto un predicado booleano $f(W,X,Y,Z)$ que será verdadero si y sólo si el símbolo Z puede aparecer en la posición j de alguna descripción instantánea dado que W, X, Y son los símbolos en las posiciones $j-1$, j y $j+1$, respectivamente de la descripción instantánea anterior. Así la fórmula será:

$$\bigwedge_{j=2}^{p(n)-1} \left(\bigvee_{\substack{W,X,Y,Z \text{ tal que} \\ f(W,X,Y,Z)}} (C_{j-1,W} \wedge C_{jX} \wedge C_{j+1,Y} \wedge D_{jZ}) \right)$$

Como el número de combinaciones para las cuales $f(W,X,Y,Z)$ es verdadero es constante, entonces podemos construir esta fórmula en $O(p^2(n))$ tiempo.

3) Tenemos que verificar que α y β definan la misma configuración. Como para cada i sólo una C_{iX} y D_{iY} son verdaderas entonces la fórmula queda como:

$$\bigwedge_{i=1}^{p(n)} \left(\bigwedge_{\substack{X \in C \\ Y \in D}} ((\neg C_{iX} \wedge D_{iY}) \wedge (C_{iX} \vee \neg D_{iY})) \right)$$

La cual puede construir en $O(p^2(n))$ tiempo.

4) Esta fórmula inicia con $Config(\alpha)$. Sea $\omega = a_1 \cdots a_n$; que α sea la configuración inicial con ω se expresa en la siguiente función :

$$C_{\substack{q_0 \\ 1a_1}} \wedge \bigwedge_{i=2}^{n} C_{ia_i} \wedge \bigwedge_{i=n+1}^{p(n)} C_{iB}$$

$C_{\substack{q_0 \\ 1a_1}}$ representa el hecho de que la cabeza lectora de M esté en la primera posición y el estado inicial q_0.

$\wedge_{i=2}^{n} C_{ia_i}$ el hecho de que el resto de la palabra considerada y $\wedge_{i=n+1}^{p(n)} C_{iB}$ el hecho de que el espacio sobrante esté en blanco.

Esta fórmula se escribe en $O(p^2(n))$ tiempo.

5) Comenzamos con $Config(\alpha)$ y verificamos que en el símbolo compuesto se encuentre un estado final.

$$\bigvee_{i=1}^{p(n)} \left(\bigwedge_{X \in C_F} C_{iX} \right)$$

Donde C_F es el conjunto de símbolos compuestos, donde el estado de el símbolo es estado final. ■

Con esta proposición como herramienta probemos el siguiente :

Lema 5 *Sea M una MT que trabaja en espacio polinomial y sea $m \in \mathbf{N} \cup \{0\}$ entonces existe una fórmula booleana cuantificada $Accsess2^m(\alpha, \beta)$ con α y β variables booleanas vectoriales la cual es verdadera si y sólo si α y β codifican configuraciones de M y M puede ir de α a β en a lo más 2^m movimientos. Además $Accsess2^m$ puede escribirse en tiempo polinomial en m.*

Demostración :
Demostraremos la fórmula por inducción.
$m = 0$
Aquí debemos de expresar los siguientes hechos.

1. Tanto α como β son codificaciones válidas.
2. Que $\alpha = \beta$ ó $\alpha \longmapsto \beta$.

Esto lo expresamos en las siguientes fórmulas booleanas:

$$Access2^0(\alpha, \beta) = Config(\alpha) \wedge Config(\beta) \wedge (Equal(\alpha, \beta) \vee Next(\alpha, \beta))$$

Esta fórmula puede escribirse en $O(p^2(n))$ tiempo (por la proposición anterior), para algún polinomio p.

Supongamos válida $Access2^{m-1}(\gamma, \delta)$, es decir, que dadas dos codificaciones válidas de M, γ y δ, $Access2^{m-1}$ es verdadera si y sólo si γ puede alcanzar a δ en a lo más 2^{m-1} pasos.

La fórmula que nos permite conocer si dadas dos codificaciones de configuraciones de M α y β, se puede llegar accesar de α a β en a lo más 2^m movimientos es la siguiente:

$$\begin{aligned} Access2^m(\alpha,\beta) \;=\; & \exists\gamma\, Config(\gamma) \wedge \\ & \forall\alpha'\forall\beta'[(Equal(\alpha',\alpha) \wedge Equal(\beta',\gamma) \vee \\ & (Equal(\alpha',\gamma) \wedge Equal(\beta',\beta)] \Longrightarrow \\ & Access2^{m-1}(\alpha',\beta')] \end{aligned}$$

La fórmula se explica con el siguiente diagrama:

$$\begin{array}{ccccccc} & & & & \alpha' & \overset{2^{m-1}}{\longmapsto} & \beta' \\ \alpha & \longmapsto & \gamma & \longmapsto & \beta & & \\ \alpha' & \overset{2^{m-1}}{\longmapsto} & \beta' & & & & \end{array}$$

Si se puede llegar de α a β en 2^m pasos entonces debe de existir una configuración γ tal que para todas las configuraciones α', β' que cumplen que $\alpha' = \alpha$ y $\beta' = \gamma$ ó $\alpha' = \gamma$ y $\beta' = \beta$ se debe de cumplir se puede llegar de α' a β' en 2^{m-1} movimientos.

Ahora bien, en número de símbolos en $Access2^m$ es $O(p^2(n))$ más el número de símbolos que utiliza $Access2^{m-1}$. Puesto que $Access2^m$ introduce $O(p^2(n))$ símbolos y el mismo razonamiento se aplica a $Access2^{m-1}$ entonces cada llamada recursiva del procedimiento ocupará m símbolos y por lo tanto $Access2^m$ ocupará $O(mp^2(n))$ símbolos. Como estos símbolos están codificados en algún alfabeto finito entonces cada uno de ellos ocupa $O(\log m + 2\log p(n)) = O(\log m + \log p(n))$. De aquí que en su conjunto todos ellos ocuparán $O(mp(n)(\log m + \log p(n)))$ símbolos. Por lo tanto la fórmula puede escribirse en tiempo polinomial. ■

Con la anterior construcción podemos entonces probar la completez de QBF en **PESPACIO**.

Teorema 6 QBF $\in$ **PESPACIO**-completo

Demostración :

Lo que falta demostrar es que QBF es **PESPACIO**-difícil. Construiremos una Fórmula Booleana Cuantificada tal que dada como entrada una codificación de alguna MT $M \in$ **PESPACIO** y una palabra ω , la fórmula sea verdadera si y sólo si $\omega \in L_M$.

Sea $M \in$ **PESPACIO** y p su polinomio que la acota en espacio. Sabemos que si $\omega \in L_M$ entonces M acepta a ω en a lo más $C^{p(n)}$ movimientos (configuraciones). Para cada entrada ω de M, con $\mid \omega \mid = n$, construyamos la siguiente fórmula:

$$Accepted(\omega) = \exists\alpha\exists\beta Initial(\alpha,\omega) \wedge Accepted(\beta) \wedge Access 2^{p(n)\log_2 C}(\alpha,\beta)$$

Por los lemas anteriores esta fórmula puede escribirse en tiempo polinomial y es verdadera si y sólo si M acepta a ω. Es decir, $\omega \in L_M$ si y sólo si $Accepted(\omega) \in QBF$.

Así esta función mapea a el cómputo de la MT M sobre ω a la fórmula booleana $Accepted(\omega)$ y es una reducción de L_M a QBF . Por lo tanto, $QBF \in$ **PESPACIO**-completo. ■

Por el teorema anterior conocemos que QBF es **PESPACIO**-completo. En la siguiente sección definiremos la clase **IP** de lenguajes.

4.4 Sistemas Interactivos de Prueba

Hasta ahora hemos manejado algoritmos (MT) en los cuales no existe la posibilidad de tener más conocimiento que el del problema mismo y el algoritmo. Este hecho no refleja necesariamente la realidad de muchos problemas, ya que en muchos de ellos es posible tener *interacción* con otros medios e información adicional. Para capturar esta idea de *comunicación* e *interacción* se definió la noción de *Sistema de Prueba Interactivo.*

Intuitivamante un *Sistema Interactivo de Prueba* para un lengueje L es un protocolo para una Máquina de Turing M_p aleatoria interactiva, con poder ilimitado de cómputo, y una Máquina de Turing M_v aleatoria interactiva acotada polinomialmente en tiempo. Estas máquinas satisfacen dos condiciones con respecto a su entrada común ω: Si $\omega \in L$ entonces con probabilidad grande, M_v se convencerá de este hecho, después de interactuar con M_p. Si $\omega \notin L$ entonces con probabilidad pequeña M_v aceptará a ω. El hecho de que M_p tenga poder ilimitado de cómputo es crucial pues permite tener información adicional y por lo tanto aumentar el poder de cómputo

del sistema. Pero así como M_p tiene poder ilimitado de cómputo, debe de existir manera de verificar de manera práctica que la información y cómputos realizados por M_p sean correctos, por esto, M_v es una máquina acotada polinomialmente en tiempo y verifica, a partir de la evidencia estadística, la información proporcionada por M_p. Los Sistemas Interactivos de Prueba aumentaron la noción estándar que se tenía de certificados de prueba en dos distintas direcciones. Primero, se permite que exista interacción (en la noción estándar de prueba sólo se dá ésta y después se verifica) pues tanto el *probador* como el *verificador* son capaces de mantener comunicación entre ellos. La Interacción se hace útil permitiendo que ambas máquinas sean aleatorias. Tanto el probador como el verificador son capaces de formular preguntas y dar respuestas a partir de sus cintas aleatorias.

Comenzemos por definir formalmente el concepto de *Máquina de Turing Interactiva.*

Definición 20 (Máquina de Turing Determinista Interactiva) .
Una Máquina de Turing Determinista Interactiva *(MTDI) es una Máquina de Turing Determinista con una cinta de entrada de sólo lectura, una cinta aleatoria de sólo lectura, una cinta de trabajo, una cinta de comunicación de sólo lectura, una cinta de comunicación de sólo escritura y una cinta de salida de sólo escritura. La cinta aleatoria contiene una cinta infinita de bits aleatorios { 0,1 }, que siguen una distribución Bernoulli($\frac{1}{2}$). Cuando se dice que la MTDI* lanza una moneda *entonces lee el siguiente bit dentro de su cinta aleatoria.*

El hecho de llamar *interactiva* a este tipo de máquina es porque es posible "conectarla" a otra MT interactiva, mediante un protocolo de comunicación.

Un *Protocolo Interactivo* es un par ordenado (M_p, M_v) de MT interactivas las cuales tienen la misma cinta de entrada, con la cinta de comunicación sólo-escritura de M_v como la cinta de comunicación de sólo-lectura de M_p y viceversa. Las dos máquinas se turnan para activarse, siendo M_v la primera en activarse. Durante su estado activo cada máquina realiza operaciones sobre sus cintas de entrada, de trabajo, de comunicación y su cinta aleatoria; al final escribe una cadena dentro de su cinta de comunicación de sólo escritura, esto es a lo que se le llama el *i-ésimo mensaje* de la máquina, si la máquina esta en su i-ésimo estado activo. En este punto M_v se desactiva, y M_p se activa, a menos de que el protocolo haya terminado (el i-ésimo mensaje de M_p se define de igual manera que el de M_v). Una máquina puede terminar el protocolo no mandando mensaje en su estado activo. M_v

acepta (*rechaza*) su entrada cambiándo a un estado de aceptación (rechazo) y terminando el protocolo. El tiempo de cómputo de M_v se define como la suma de los cómputos de M_v durante sus estados activos. De igual forma el tiempo de cómputo de M_p se define como la suma de sus cómputos durante sus periódos activos. Requerimos que el tiempo de cómputo de M_v esté acotado polinomialmente en la longitud de la cadena de entrada. M_p tiene poder ilimitado de cómputo y el único recurso en que se acota a M_p es que la longitud de los mensajes que manda a M_v estén acotados polinomialmente en la longitud de la cadena de entrada; puede utilizar tanto tiempo y espacio como requiera para computar el mensaje que enviará. Una *cominucación* es un mensaje de M_v a M_p o viceversa. La *i-ésima vuelta* del protocolo es el i-ésimo movimiento de M_v a M_p junto con la respuesta de M_p a M_v. El *texto o história del cómputo* es la sucesión $h_n = \{v_1, p_1, \cdots, v_n, p_n\}$, donde v_i (respectivamente p_i) denota el mensaje mandado de M_v a M_p (de M_p a M_v) en la i-ésima vuelta. Si $p_n = \epsilon$ entonces M_p detiene el cómputo en la i-ésima vuelta.

Como las dos máquinas de Turing interactivas son aleatorias, podemos definir un espacio de probabilidad sobre el conjunto de textos de todos los posibles cómputos de (M_p,M_v) sobre alguna entrada ω, de tal forma que la probabilidad de cada cómputo de (M_p,M_v) sobre la entrada ω se toma sobre los bits aleatorios de las máquinas M_p y M_v.

A partir de todo lo mencionado anteriormente, podemos dar la siguiente definición:

Definición 21 *Dado un lenguaje $L \subseteq \{0,1\}^*$, decimos que L tiene un* Sistema de Prueba Interactivo *si $\exists M_v$ MTI de tiempo polinomial tal que*

1. *$\exists M_p$ MTI tal que (M_p,M_v) es un protocolo interactivo y $\forall \omega \in L$ con $\mid \omega \mid$ suficientemente grande, la probabilidad de que M_v acepte a ω es mayor que $\frac{2}{3}$.*

2. *$\forall M_p$ MTI tal que (M_p,M_v) formen un protocolo interactivo y para toda $\omega \in (\{0,1\}^* \setminus L)$, con $\mid \omega \mid$ suficientemente grande, la probabilidad de que M_v acepte a ω es menor a $\frac{1}{3}$.*

Donde las probabilidades se toman sobre los bits aleatorios de M_p y M_v.

La primera condición se le llama de *completez* y a la segunda se le llama de *solidez*.

Se demuestra en [15] que la elección de las constantes $\frac{1}{3}$ y $\frac{2}{3}$ no afecta al conjunto de lenguajes que pueden ser reconocidos por los Sistemas Interactivos de Prueba. La elección de las constantes $\frac{1}{3}$ y $\frac{2}{3}$ es sólo por simplicidad; cualquier valor acotado por $\frac{1}{2}$ cumplirá con las condiciones de completez y solidez. Si realizamos ejecuciones repetidas del Sistema Interactivo de Prueba, se pueden siempre cumplir requerimientos tan estrictos como $1 - (\frac{1}{2^{|\omega|}})$ para la condición de completez y $\frac{1}{2^{|\omega|}}$, cuando ω es la cadena de entrada.

Si tenemos un protocolo interactivo (M_p, M_v) llamaremos a M_p *probador* y a M_v *verificador.*

Definición 22 *Dada* $q : \mathbf{N} \longrightarrow \mathbf{N}$ *no decreciente decimos que* (M_p, M_v) *es un* sistema de prueba q-vuelta *para* $L \subseteq \{0,1\}^*$, *si* $\forall \omega \in L$ *la longitud de la história del cómputo de* (M_p, M_v) *sobre* ω *está acotado por* $q(|\ \omega\ |)$.

Ahora definiremos la clase de lenguajes **IP**.

Definición 23 *Definimos* $\mathbf{IP}[q(n)]$ *como la clase de lenguajes para los cuales existe un sistema de prueba* $q(n)$*-vuelta. En general definimos a* **IP** *como:*

$$\mathbf{IP} = \bigcup_{k \geq 0} \mathbf{IP}[n^k]$$

Ahora presentaremos un ejemplo de sistema interactivo de prueba.

Ejemplo. Definamos el problema de *gráficas no-isomórfas* por :

$$NONISO = \{(G_0, G_1) \mid G_0 \ncong G_1\}$$

donde el símbolo $\ncong$ denota no isomorfismo entre las gráficas.

Dadas como entrada dos gráficas $G_0(V_0, E_0)$ y $G_1(V_1, E_1)$, con $n = |\ V_0\ | = |\ V_1\ |$. Vamos a suponer en el protocolo que las gráficas tienen el mismo número de vértices, pués en el caso en el que las gráficas tienen distinto número de vértices el problema es trivial. Consideraremos el siguiente protocolo :

1. M_v lee la entrada y escoge de su cinta aleatoria n enteros $c_i \in \{0,1\}$ para $1 \leq i \leq n$. Para cada c_i, M_v genera una gráfica $H_i(V, E_i)$ la cual es isomorfa a G_{c_i}, ésto lo hace escogiendo aleatoriamente una permutación $\pi : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ y aplicando esta permutación a G_{c_i}. Al terminar, M_v manda a M_p las gráficas $H_i(V, E_i)$, $1 \leq i \leq n$.

2. El probador lee la entrada y prueba si G_0 es isomorfa a G_1. Esto lo hace calculando las permutaciones $\pi : \{1,\ldots,n\} \longrightarrow \{1,\ldots,n\}$, luego las aplica a G_0 y verificando después si la permutación π aplicada a G_0 hace que G_0 sea igual a G_1. (Esto puede computarlo pues tiene poder ilimitado). En caso de que no lo sean, el probador lee el mensaje dejado por el verificador y para cada $1 \leq i \leq n$ M_p deja un entero α_i definido de la siguiente manera:

$$\alpha_i = \begin{cases} 0 & H_i \cong G_0 \\ 1 & H_i \cong G_1 \end{cases}$$

Si $G_0 \cong G_1$ entonces para cada i, M_p escoge $\alpha_i \in \{0,1\}$ con probabilidad $\frac{1}{2}$. En ambos casos, cuando M_p ha producido $\{\alpha_1,\cdots,\alpha_n\}$ los manda a M_v.

3. M_v lee el mensaje $\{\alpha_1,\cdots,\alpha_n\}$ del probador. M_v compara α_i con c_i. Si para algún i se tiene que $\alpha_i \neq c_i$ entonces suceden una de las siguientes dos cosas: *i*) las gráficas son isomorfas ó *ii*) el probador ha cometido una equivocación; en ambos casos M_v rechaza la entrada. De otra manera, M_v acepta la entrada (es decir, M_v verificó que $G_0 \ncong G_1$).

Ya definido el protocolo, demostremos que este protocolo forma una prueba interactiva para *NONISO*. Para probar ésto debemos demostrar que la prueba interactiva presentada cumple con las condiciones de la Definición 21.

Sea (G_0,G_1) que se dan como entrada al sistema interactivo. Supongamos que $G_0 \ncong G_1$ (es decir, $(G_0,G_1) \in NONISO$). El paso 1 del protocolo se realiza siempre sin importar si $G_0 \cong G_1$ o no lo son. El paso 2 inicia cuando M_p prueba si $G_0 \cong G_1$ (esto lo hace calculando las permutaciones $\pi : \{1,\ldots,n\} \longrightarrow \{1,\ldots,n\}$, luego las aplica a G_0 y verifica después si la permutación π aplicada a G_0 hace que G_0 sea igual a G_1) como por hipótesis, $G_0 \ncong G_1$ entonces (por la definición del protocolo), M_p prueba para cada $1 \leq i \leq n$ si H_i es isomorfa a G_0 ó a G_1 y manda a M_v la cadena $\alpha_1 \ldots \alpha_n$. En el paso 3 M_v lee $\alpha_1 \ldots \alpha_n$ y los compara con las c_i. Si $c_i = \alpha_i$ $1 \leq i \leq n$ entonces se tiene que $G_0 \ncong G_1$. Como en este caso toda la información que se maneja es determinista entonces si M_v encuentra que $c_i = \alpha_i$, aceptará con probabilidad 1. Ahora supongamos que $G_0 \cong G_1$ (es decir, $(G_0,G_1) \notin NONISO$), el protocolo ejecuta el paso 1 normalmente. Al llegar al paso 2, como encuentra que $G_0 \cong G_1$ (por hipótesis) entonces

$H_i \cong G_0$ y $H_i \cong G_1$ de aquí que cualquier probador no será capaz de decidir si $\alpha_i = 0$ ó $\alpha_i = 1$. Entonces M_p decide aleatoriamente si $\alpha_i = 0$ ó $\alpha_i = 1$, esta elección la realiza aleatoriamete con probabilidad $\frac{1}{2}$ (eligiendo un bit aleatorio de su cinta aleatoria), para cada evento. En el paso tres, al leer M_v el mensaje $\alpha_1 \ldots \alpha_n$ y comparar si $\alpha_i = c_i$ entonces decide con probabilidad de error es igual a $\frac{1}{2}$ el hecho de que $\alpha_i \neq c_i$. Así la probabilidad de que M_v acepte la entrada $(G_0, G_1) \notin NONISO$ es igual a la probabilidad de que $\alpha_i = c_i$ $1 \leq i \leq n$, la cual es $\frac{1}{2^n} < \frac{1}{3}$. En ambos casos ($G_0 \cong G_1$ ó $G_0 \ncong G_1$) se mantienen las condiciones de la Definición 21 y por lo tanto el protocolo presentado es un Sistema Interactivo de Prueba para $NONISO$.

Por lo tanto, por definición $NONISO \in IP[2]$.

Este es un protocolo sencillo que ejemplifica la forma en que trabajan los Sistemas Interactivos de Prueba. En el siguiente Capítulo se construirá un Sistema de Prueba Interactivo para QBF.

Capítulo 5

IP = PESPACIO

En general si tenemos una clase de lenguajes $\mathcal{C}$ y conocemos que existe algún lenguaje $L \in \mathcal{C}$-completo; si podemos asegurar que $L \in \mathcal{D}$ (otra clase de lenguajes) entonces $\mathcal{C} \subseteq \mathcal{D}$. Este razonamiento se utilizará para demostrar que **PESPACIO** $\subseteq$ **IP**. La contención en el otro sentido **IP** $\subseteq$ **PESPACIO** está dada en [5]. La demostración se basa, esencialmente en que si un lenguaje está en **IP** entonces puede ser aceptado por una Máquina determinista que trabaje en espacio polinomial y que explore el árbol de todas las posibles interacciones de su Sistema Interactivo de Prueba. El Sistema Interactivo de Prueba presentado está basado en [19].

5.1 Aritmetización de *QBF*

Si tenemos que $\mathcal{F}$ es una fórmula booleana cuantificada entonces si queremos decidir si $\mathcal{F} \in QBF$ entonces este problema se puede trasladar a un problema de evaluación de una fórmula aritmética. Para trasladar este problema usamos las siguientes transformaciones:

1. Reemplazar cada variable booleana x_i por una nueva variable aritmética $z_i \in \mathbf{Z}$ y trasladamos los valores lógicos *true* $\rightarrow 1$ y *false* $\rightarrow 0$.

2. Reemplazar $\overline{x_i} \rightarrow (1 - z_i)$, $x_i \wedge x_j \rightarrow z_i z_j$, $x_i \vee x_j \rightarrow 1 - (1 - z_i)(1 - z_j) := z_i * z_j$

3. Reemplazar los cuantificadores de la siguiente manera:

   - $\forall x\, B(x) \rightarrow b(0)b(1) := \prod_z b(z)$

- $\exists x\, B(x) \rightarrow b(0) + b(1) - b(0)b(1) := \coprod_z b(z)$
  Donde $b()$ es la Fórmula Booleana sin cuantificadores a la cual se le ha aplicado 1. y 2.

El objetivo de las anteriores transformaciones es preservar el valor lógico de cualquier Fórmula Booleana Cuantificada ($false \longrightarrow 0$, $true \longrightarrow 1$) en su valor aritmético. En el inciso dos se realizan las transformaciones para fórmulas booleanas. Una fórmula booleana se puede satisfacer si y solo si su equivalente aritmético evalúa a 1. El inciso tres realiza las transformaciones sintácticas para las fórmulas booleanas pero ya cuantificadas. Estas definiciones se siguen inmediatemante del inciso 2 y de la forma de evaluar los cuantificadores (Def. 7). Cuando a una fórmula booleana cuantificada $\mathcal{F}$ le aplicamos las anteriores transformaciones sintácticas y obtenemos $A_{\mathcal{F}}$, entonces decimos que a $\mathcal{F}$ le aplicamos su proceso de *aritmetización.*

Ejemplo. Sea $\mathcal{F} = \forall x_1 \exists x_2 \forall x_3 (x_1 \wedge x_2) \vee x_3$, se tiene que $V(\mathcal{F}) = false$.
Primero, $(x_1 \wedge x_2) \vee x_3 \rightarrow 1 - (1 - z_1 z_2)(1 - z_3) = b(z_1, z_2, z_3)$. Denotaremos la forma aritmética de $\mathcal{F}$ por $A_{\mathcal{F}}$, entonces

$$A_{\mathcal{F}} = \prod_{z_1} \coprod_{z_2} \prod_{z_3} b(z_1, z_2, z_3)$$

La cual es igual a 0, evaluando la fórmula según las reglas dadas anteriormente.

Observamos que si una variable booleana no se cuantifica, entonces la fórmula no puede ser evaluada a *true* ó *false*. Por esto las fórmulas en las cuales todas sus variables están cuantificadas reciben una denominación especial.

Definición 24 *Sea $\mathcal{F}$ una fórmula booleana cuantificada . Si todas las variables de $\mathcal{F}$ se encuentran cuantificadas en $\mathcal{F}$ entonces a $\mathcal{F}$ se le llama* cerrada. *De otra manera a $\mathcal{F}$ se le llama* abierta.

Por el ejemplo anterior, podemos intuir que la transformación de una fórmula booleana cuantificada a su forma aritmética nos proporciona una manera de decidir acerca de las fórmulas booleanas cuantificadas. Este resultado queda plasmado en el siguiente teorema.

Teorema 7 *Sea $\mathcal{F}$ una fórmula booleana cuantificada cerrada, entonces el valor booleano de $\mathcal{F}$ coincide con su correspondiente valor aritmético $A_{\mathcal{F}}$.*

Demostración :

Se demostrará por inducción sobre la estructura de $\mathcal{F}$. Como primer paso de la inducción demostremos que las transformaciones sintácticas definidas son correctas.

Sean x_1 y x_2 variables boolenas y z_1 y z_2 sus variables enteras asociadas. Por definición, $x_1 = true$ si y sólo si $z_1 = 1$ y $x_1 = false$ si y sólo si $z_1 = 0$. Se hará la verificación para el caso de la disyunción; la verificación para la conjunción y negación son totalmente análogas. Sea, $x_1 \vee x_2 = false$ ssi $x_1 = false = x_2$, de aquí que $z_1 = 0 = z_2$. La transformación definida es $x_1 \vee x_2 \rightarrow 1 - (1 - z_1)(1 - z_2) = 1 - (1 - 0)(1 - 0) = 0$. Si $x_1 \vee x_2 = true$ entonces al menos alguna de las variables es verdadera; supongamos que $x_1 = true$, entonces como $x_1 \vee x_2 \rightarrow 1 - (1 - z_1)(1 - z_2) = 1 - (1 - 1)(1 - z_2) = 1$. Por lo tanto, el valor booleano de la disyunción queda reflejado en su forma aritmética.

La tercera regla proporciona la transformación cuando tenemos cuantificadores. Si tenemos que $\mathcal{F} = \forall x\, B(x)$ donde $B(x)$ está libre de cuantificadores entonces $V(\mathcal{F}) = B(false) \wedge B(true) \rightarrow b(0)b(1)$ donde $b(z)$ es el polinomio correspondiente a la fórmula booleana (libre de cuantificadores) B. Si $\mathcal{F} = true$ entonces $B(false) = true = B(true)$; de aquí que $b(0) = 1 = B(true)$ lo cual implica que $A_{\mathcal{F}} = b(0)b(1) = 1$. Si $\mathcal{F} = false$ entonces al menos una de las fórmulas $B(false)$ ó $B(true)$ es $false$; de aquí que al menos una de $b(0)$ o $b(1)$ es iguales a 0. Por lo tanto, $A_{\mathcal{F}} = b(0)b(1) = 0$. El caso para cuando $\mathcal{F} = \exists x\, B(x)$ es totalmente análogo.

Así queda establecido que la transformación definida cumple el teorema (base de la inducción).

Supongamos que tenemos una fórmula booleana cuantificada cerrada $\mathcal{F}$ con $n+1$ variables. Entonces $\mathcal{F} = K_{n+1}x_{n+1}K_n x_n \cdots K_1 x_1\, B(x_{n+1}, \ldots, x_1)$ donde $K_i \in \{\forall, \exists\}$ y x_i son variables booleanas y $B(\cdots)$ es libre de cuantificadores. Como hipótesis de inducción supongamos que toda fórmula booleana cerrada con n variables cumple nuestro teorema; queremos demostrar que la fórmula con $n+1$ variables también lo cumplen.

Supongamos que $K_{n+1} = \forall$, entonces

$$\mathcal{F} = \forall x_{n+1} K_n x_n \cdots K_1 x_1\, B(x_{n+1}, x_n, \ldots, x_1) = \forall x_{n+1} F'$$

De aquí F' es una fórmula booleana cuantificada abierta y su valor depende del valor de x_{n+1}, es decir, $F' = F'(x_{n+1})$. De todo esto se sigue:

$$\begin{aligned} \mathcal{F} &= \forall x_{n+1} K_n x_n \cdots K_1 x_1 B(x_{n+1}, x_n, \ldots, x_1) \\ &= \forall x_{n+1} F'(x_{n+1}) \\ &= F'(false) \wedge F'(true) \end{aligned}$$

la cual puede evaluarse por hipótesis de inducción (pues $f'(0)f'(1)$ son fórmula booleana cuantificada con n variables.

El caso para cuando $K_{n+1} = \exists$ es totalmete análogo. ■

El teorema anterior nos proporciona una posible manera de evaluar una fórmula booleana cuantificada $\mathcal{F}$ (es decir saber si $\mathcal{F} \in QBF$) vía la evaluación de su forma aritmética . El proceso no es tan sencillo como pudiera parecer, veamos el siguiente ejemplo.

Ejemplo. Analizar la siguiente fórmula booleana cuantificada vía su aritmetización.

$$F = \forall x_n \cdots \forall x_1 \left[(x_1 \wedge \cdots \wedge x_{n-1}) \vee x_n\right]$$

La aritmetización de la fórmula booleana $(x_1 \wedge \cdots \wedge x_{n-1}) \vee x_n$ es:

$$b = 1 - (1 - z_n)(1 - \prod_{i=1}^{n-1} z_i) = z_n + (1 - z_n) \prod_{i=1}^{n} z_i$$

Analizemos la forma en la que el grado de b crece al momento de su evaluación aritmética.

$$\begin{aligned} A_F &= \prod_{z_n} \cdots \prod_{z_2} \prod_{z_1} \left[z_n + (1 - z_n) \prod_{i=1}^{n-1} z_i \right] \quad grado(b) = 2^0 \\ &= \prod_{z_n} \cdots \prod_{z_3} \prod_{z_2} [z_n] \left[z_n + (1 - z_n) \prod_{i=2}^{n-1} z_i \right] \quad grado(b) = 2^1 \\ &= \prod_{z_n} \cdots \prod_{z_4} \prod_{z_3} [z_n^2] \left[z_n^2 + (1 - z_n) z_n \prod_{i=3}^{n-1} z_i \right] \quad grado(b) = 2^2 \end{aligned}$$

Por simple inspección de los valores de los grados de b se nota que el proceso de evaluación de la fórmula en la $n-1$ evaluación se tendrá que el polinomio b será de grado 2^n en z_n. Este es un serio inconveniente pués dentro del protocolo el verificador no podría leer este polinomio (evaluado para algún valor aleatorio, necesario para verificar probabilísticamente el cómputo del probador), es decir, la longitud de este número en algún alfabeto binario es exponencial; esto es cierto pués si tenemos a un número a representado en base binaria, entonces el número de bits que ocupará en su representación binaria es $\lceil \log_2 a \rceil + 1$. Si tenemos que $a = z^{2^n}$ entonces la longitud de su representación en algún alfabeto binario será de $\lceil \log_2 x^{2^n} \rceil + 1 = \lceil 2^n \frac{\ln x}{\ln 2} \rceil + 1$. Para evitar este problema se introduce un operador que reduce el grado de los polinomios, pero que deja intacto el valor aritmético de estos, tomando a sus variables en el conjunto $\{0,1\}$.

Definición 25 *Sea* $P(x_1,\ldots,x_n)$ *un polinomio. Entonces*

$$(Rx_1\,P)(x_1,\ldots,x_n) \;=\; P(x_1,\ldots,x_n) \bmod {}^1(x_1^2 - x_1)$$

Operativamente se están reemplazando todas las ocurrencias de $x^n, n > 1$ por x. Esto es claro, pués si consideramos que si tenemos algún polinomio $P'(x_1,\ldots,x_n)$ entonces si dejamos fijas a las variables $x_2,\ldots,x_n$ y tomamos a $P'(x_1,\ldots,x_n) = P(x_1)$ tenemos que este polinomio puede verse como:

$$P(x_1) = \sum_{i=1}^{n} a_i x_1^i.$$

Ahora bien, puesto que

$$\begin{aligned} P(x_1) \;&=\; [a_n x_1^{n-2} + (a_n + a_{n-1})x_1^{n-3} + \cdots + (a_n + \cdots + a_2)](x_1^2 - x_1) \\ &+\; (a_1 + \cdots + a_n)x_1 \end{aligned}$$

entonces por definición

$$(Rx_1P)(x_1) = (a_1 + \cdots + a_n)x_1.$$

[1] La definición del operador *mod* en un anillo $\mathcal{A}$ es:

$$a \bmod b = r \quad \Leftrightarrow \quad a = r + bq, \quad q \in \mathcal{A}.$$

entonces $a \bmod 0 = a$.

Es decir, tomamos todas las ocurrencias x_1^n, $n > 1$ y las sustituimos por x_1. Hay que notar que los coeficientes de x_1 no se verán afectados durante este proceso (es decir, Rx_1P sólo afecta a x_1).

Recordemos que nuestro objetivo es evaluar los polinomios obtenidos en el Teorema 7. Los polinomios P y $P \bmod (x^2 - x)$ evalúan lo mismo cuando $x \in \{0,1\}$.

Proposición 5
Sean $P(x_1,\ldots,x_n)$ un polinomio y $Q(x_1,\ldots,x_n) = (Rx_1P)(x_1,\ldots,x_n)$ entonces

$$P(a_1,\cdots,a_n) = Q(a_1,\cdots,a_n) \quad para \quad a_1 \in \{0,1\}$$

Demostración:
Por definición tenemos que:

$$\begin{aligned} Q(x_1,\cdots,x_n) &= (Rx_1P)(x_1,\cdots,x_n) \\ &= P(x_1,\cdots,x_n) \bmod (x_1^2 - x_1) \end{aligned}$$

Esto sucede si y sólo si existe un polinimio $R(x_1,\ldots,x_n)$ tal que

$$P(x_1,\ldots,x_n) = Q(x_1,\ldots,x_n) + (x_1^2 - x_1)R(x_1,\ldots,x_n)$$

y esto si y sólo si

$$Q(x_1,\cdots,x_n) - P(x_1,\cdots,x_n) = R(x_1,\cdots,x_n)(x_1^2 - x_1)$$

En ambos caso si hacemos $x_1 = 0$ ó $x_1 = 1$ entonces $P = Q$. ■

Hay que tener muy presente esta proposición al momento del análisis del protocolo. Otro truco para reducir el órden de las operaciones es realizar los cálculos *mod* algún primo p. Para garantizar que este número existe y que no afecta nuestra decisión debemos mostrar el siguiente resultado.

Teorema 8 *Sea $\mathcal{F}$ una fórmula booleana cuantificada cerrada de longitud n. Entonces $\exists\, p$ primo tal que p es de longitud polinomial en n y $A_{\mathcal{F}} \neq 0 \bmod p$ si y sólo si $\mathcal{F} = true$*

Demostración :

Tenemos una proposición lógica $(p \Rightarrow q \Leftrightarrow r)$. La demostración se realizará por contradicción $(p \wedge \neg q \Leftrightarrow r$ con lo cual se debe llegar a una contradicción).

Sea $\mathcal{F}$ una fórmula booleana cuantificada y $A_{\mathcal{F}}$ su forma aritmética. Supongamos que $A_{\mathcal{F}} = 0 \pmod p$ $\forall p$ *primo* tal que $\mid p \mid \leq q(n)$, con q un polinomio, entonces:

$$A_{\mathcal{F}} = 0 \pmod{\prod_{|p| \leq q(n)} p}$$

como $\mid p \mid = \log_2 p + 1$ (número de bits necesarios para representar a p en un alfabeto binario) entonces

$$A_{\mathcal{F}} = 0 \pmod{\prod_{p \leq 2^{q(n)-1}} p}$$

lo cual implica por definición que

$$A_{\mathcal{F}} = K \prod_{p \leq 2^{q(n)-1}} p$$

como $\mathcal{F} = true$ entonces $A_{\mathcal{F}} \neq 0$ de aquí que K sea distinto de cero. Por el Teorema de los Números Primos [9]

$$\prod_{p \leq 2^{q(n)-1}} p = O(2^{q(n)-1})$$

pero

$$\begin{aligned} \mid \prod_{p \leq 2^{q(n)-1}} p \mid &= \log_2 \prod_{p \leq 2^{q(n)-1}} p - 1 \\ &= \log_2 2 \prod_{p \leq 2^{q(n)-1}} p \\ &= y \end{aligned}$$

donde

$$y = 2^2 \prod_{p \leq 2^{q(n)-1}} p$$

es decir, la longitud de $\prod_{|p|\leq 2^{q(n)}} p$ es de orden exponencial, lo cual es una contradicción con la hipótesis de que $A_{\mathcal{F}}$ es cero para, a lo mas, los números de tamaño polinomial. ■

Con todos los elementos expuestos hasta ahora, estamos en condiciones de definir el protocolo interactivo para *QBF*.

5.2 Protocolo interactivo para *QBF*

Definición 26 *Sea $\mathcal{F}$ una fórmula booleana cuantificada cerrada y $A_{\mathcal{F}}$ su aritmetización. La* forma funcional *de $\mathcal{F}$ queda definida quitando el símbolo $\coprod$ ó $\prod$ más a la izquierda de $A_{\mathcal{F}}$.*

Ejemplo. Sea $\mathcal{F} = \forall x_1 \exists x_2 \forall x_3 [(x_1 \wedge x_2) \vee x_3]$ entonces:

$$A_{\mathcal{F}} = \prod_{z_1} \coprod_{z_2} \prod_{z_3} [1 - (1 - z_1 z_2)(1 - z_3)]$$

entonces la forma funcional de $\mathcal{F}$ es :

$$\begin{aligned} A'_{\mathcal{F}} &= \coprod_{z_2} \prod_{z_3} [1 - (1 - z_1 z_2)(1 - z_3)] = z_1 \\ &= \coprod_{z_2} [1 - (1 - z_1 z_2)] \\ &= z_1 \end{aligned}$$

Notemos que la forma funcional de alguna fórmula booleana cuantificada se reduce a un polinomio de una sola variable. De aquí que si conocemos la forma funcional $A'_{\mathcal{F}}$ entonces el valor booleano de $\mathcal{F}$ puede determinarse a partir de la variable de la forma funcional.

Teorema 9 *El problema* QBF *tiene un Sistema Interactivo de Prueba que trabaja en tiempo polinomial.*

Demostración :

Sea $\mathcal{F} = K_1x_1 \cdots K_nx_n\, B(x_1, \ldots, x_n)$ una fórmula booleana cuantificada , con $K_i \in \{\forall, \exists\}$, B una fórmula booleana libre de cuantificadores y $A_{\mathcal{F}} = k_1z_1 \cdots k_nz_n\, b(z_1, \ldots, z_n)$ su aritmetización. Aplicaremos a $A_{\mathcal{F}}$ operaciones R en el siguiente órden:

$$k_1z_1Rz_1k_2z_2Rz_1Rz_2k_3z_3 \cdots k_nz_nRz_1 \cdots Rz_1\, b(z_1, \ldots, z_n)$$

esta transformación la genera M_v como primer paso de la prueba y se la dá a conocer a M_p.

Después de la aplicación consecutiva de las $Rx_i'^s$ el grado de las variables involucradas se reduce a 1. El siguiente cuantificador a lo mas duplica el grado del resto de las variables; de aquí que el grado del polinomio nunca sea mayor a 2 en cualquier variable. Ahora presentemos el $IP - protocolo$ para QBF.

1. M_v pregunta a M_p por el primo p mediante el cual realizará los cálculos.
2. M_p regresa el primo p.
3. Si $A_{\mathcal{F}}$ es constante M_v acepta la fórmula sino M_v pregunta a M_p por el valor de $A_{\mathcal{F}}$.
4. M_p regresa el valor de $A_{\mathcal{F}} \longrightarrow r_0$
5. Si el primer símbolo de $A_{\mathcal{F}} \neq \prod o \coprod$ entonces ir a 9.
6. M_v pregunta por la forma funcional de $A_{\mathcal{F}}$,

$$Rz_1k_2z_2 \cdots k_nz_nRz_1 \cdots Rz_n\, b(z_1, \ldots, z_n)$$

7. M_p regresa $p_1(z_1)$ (donde $p_1(z_1)$ es la forma funcional de $A_{\mathcal{F}}$ con el operador Rz_1).
8. M_v verifica que las siguientes condiciones se cumplan.
   - Si $k_1 = \coprod$ entonces verifica $p_1(0) + p_1(1) - p_1(0)p_1(1) = r_0$
   - Si $k_1 = \prod$ entonces verifica $p_1(0)p_1(1) = r_0$

   Es decir la información proporcionada es correcta.
9. Para asegurarse que la información es correcta, M_v genera α aleatorio.

10. Para confirmar que $p_1(z_1) = Rz_1k_2z_2\cdots k_nz_nRz_1\cdots Rz_n\, b(z_1,\ldots,z_n)$ pregunta por el polinomio $k_2z_2\cdots k_nz_nRz_1\cdots Rz_1\, b(z_1,\ldots,z_n)$.

11. M_p regresa $p_2(z_1)$ (donde $p_2(z_1)$ es la forma funcional de $A_{\mathcal{F}}$ proporcionado por M_p a M_v sin el operador Rz_1).

12. M_v verifica que

$$p_2(z_1)\, mod\,(z_1^2 - z_1)\,|_\alpha = p_1(\alpha)$$

13. M_v para asegurarse que p_2 es correcto genera β aleatorio y calcula $p_2(\beta)$.

14. Ir a 3 quitando a $A_{\mathcal{F}}$ su primer operador.

En caso de que en algún momento no se verificara alguna de las condiciones que checa este protocolo entonces rechazamos el valor proporcionado por M_p.

Ahora lo que hay que demostrar es que el protocolo anterior es realmente un Sistema de Prueba Interactivo, para esto hay que verificar si:

$$\Pr[M_v\ acepte\ a\ \omega \mid \omega \notin L] \leq \frac{1}{3}$$

y

$$\Pr[M_v\ acepte\ a\ \omega \mid \omega \in L] \leq \frac{2}{3}.$$

Observamos que:

$$\Pr[M_v\ acepte\ a\ \omega \mid \omega \notin L] = \Pr[M_p\ no\ engañe\ a\ M_v]$$

y que

$$\begin{aligned}\Pr[M_v\ acepte\ a\ \omega \mid \omega \in L] &= \Pr[M_p\ no\ engañe\ a\ M_v] \\ &= 1 - \Pr[M_p\ engañe\ a\ M_v]\end{aligned}$$

Aquí las probabilidades dependen exclusivamente de $\Pr[M_p\ engañe\ a\ M_v]$; por lo tanto, debemos analizar esta probabilidad.

Durante el protocolo se realiza una vuelta por cada símbolo $\prod$, $\coprod$ y R, así el número de vueltas es $O(\mid \mathcal{F} \mid^2)$. Ahora bien M_p puede engañar a M_v dandole un polinomio que no corresponde a alguna forma funcional pedida. Como el grado de estos polinomios es menor o igual a 2 entonces por el Teorema de Interpolación, M_p puede proporcionar un polinomio que se intersecte con el correcto en $\mathbf{Z}_p$ (todos los cálculos se realizan en $\mathbf{Z}_p$ y por lo tanto los polinomios toman valores en $\mathbf{Z}_p$) con probabilidad menor o igual a $\frac{2}{p}$. También M_v puede ser engañado al hacer su elección aleatoria para verificar la veracidad de las formas funcionales proporcionadas; como estamos trabajando en $\mathbf{Z}_p$ la probabilidad de que sea engañado al hacer esta elección es $\frac{1}{p}$. Así la probabilidad de que M_p pueda engañar a M_v es menor o igual a $\frac{3|\mathcal{F}|^2}{p}$ (en el número total de vueltas). Así si escogemos $p > 3 \mid \mathcal{F} \mid^3$ entonces aseguramos que la probabilidad de que M_p engañe a M_v sea menor a $\frac{1}{|\mathcal{F}|}$ que tiende a cero cuando la longitud de $\mathcal{F}$ tiende a infinito. También se tiene que la longitud de este primo es de longitud polinomial. Por lo tanto, se tiene que

$$\Pr[M_p \textit{ engañe a } M_v] < \frac{1}{|\mathcal{F}|}$$

de aquí que:

$$\begin{aligned} \Pr[M_v \textit{ acepte a } \omega \mid \omega \notin L] &= \Pr[M_p \textit{enagñea} M_v] \\ &< \frac{1}{|\mathcal{F}|} \end{aligned}$$

y también

$$\begin{aligned} \Pr[M_v \textit{ acepte a } \omega \mid \omega \in L] &= 1 - \Pr[M_p \textit{ engañe a } M_v] \\ &= 1 - \frac{1}{|\mathcal{F}|} \\ &> \frac{2}{3} \end{aligned}$$

lo cual sucede cuando $\mid \mathcal{F} \mid$ es suficientemente grande. Por lo tanto el protocolo presentado sí es un Sistema de Prueba Interactivo para QBF .■

Ahora bien, como sabemos por el Teorema 6 del Capítulo 2 que $QBF \in$ **PESPACIO** – *completo* y por el Teorema anterior que $QBF \in$ **IP** en-

tonces cualquier lenguaje $L \in$ **PESPACIO** tendrá un Sistema Intreractivo de Prueba de tiempo polinomial y por lo tanto estará en **IP**.

Con este teorema hemos demostrado que **PESPACIO** $\subseteq$ **IP**. Ahora presentaremos un ejemplo de cómo funciona este protocolo.

Ejemplo. Sea $\mathcal{F} = \exists x_1 \forall x_2 \exists x_3 [x_1 \vee (x_2 \wedge x_3)]$, $V(\mathcal{F}) = true$, entonces:

$$A_{\mathcal{F}} = \coprod_{z_1} \prod_{z_2} \coprod_{z_3} [z_1 + z_2 z_3 - z_1 z_2 z_3]$$

la fórmula que M_v genera es :

$$A_{\mathcal{F}} = \coprod_{z_1} Rz_1 \prod_{z_2} Rz_1 Rz_2 \coprod_{z_3} Rz_1 Rz_2 Rz_3 [z_1 + z_2 z_3 - z_1 z_2 z_3]$$

M_p proporciona el valor de $1 = A_{\mathcal{F}} \rightarrow r_0 = 1$ y $p_1(z_1) = z_1$. M_v checa $p_1(0) + p_1(0) - p_1(0)p_1(1) = 1$ y genera $\alpha = 3$. M_p proporciona $p_2(z_1) = z_1$, M_v checa que $p_2(z_1) \bmod (z_1^2 - z_1) \mid_3 = 3 = p_1(z_1)$, genera $\beta = 2$ y calcula $p_2(\beta) = 2$. Aquí lo siguiente es verificar los operadores R pero como la forma funcional no tiene grados mayores a uno entonces estos operadores no afectan la fórmula, por esta razón no los verificamos en este ejemplo, aunque dentro del protocolo siguen verificandose.

M_v pregunta por la siguiente fórmula:

$$A_{\mathcal{F}} = \coprod_{z_2} Rz_1 Rz_2 \coprod_{z_3} Rz_1 Rz_2 Rz_3 [2 + 2z_3 - 2z_2 z_3]$$

M_p proporciona el valor de $1 = A_{\mathcal{F}} \rightarrow r_0$ y $p_1(z_1) = z_1$. M_v checa $p_1(0) + p_1(0) - p_1(0)p_1(1) = 1$ y genera $\alpha = 3$. M_p proporciona $p_2(z_1) = z_1$, M_v checa que $p_2(z_1) \bmod (z_1^2 - z_1) \mid_3 = 3 = p_1(3)$, genera $\beta = 2$ y calcula $p_2(\beta) = 2$. Aquí lo siguiente es verificar los operadores R pero como la forma funcional otra vez no tiene grados mayores a uno entonces estos operadores no afectan la fórmula.

M_v pregunta por la siguiente fórmula:

$$A_{\mathcal{F}} = \prod_{z_2} Rz_1 Rz_2 \coprod_{z_3} Rz_1 Rz_2 Rz_3 [2 + 2z_3 - 2z_2 z_3]$$

M_p proporciona el valor de $32 = A_{\mathcal{F}} \rightarrow r_0$ y $p_1(z_2) = 8 - 4z_2$. M_v checa $p_1(0)p_1(1) = 32$ y genera $\alpha = 2$. M_p proporciona $p_2(z_2) = 8 - 4z_2$, M_v checa que $p_2(z_2) \, mod \, (z_2^2 - z_2) \mid_2 = 0 = p_1(2)$, genera $\beta = 3$ y calcula $p_2(\beta) = -4$.

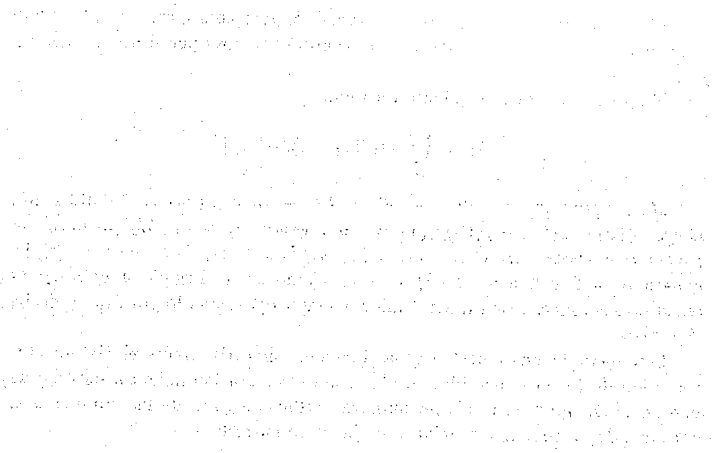
Aquí lo siguiente es verificar los operadores R pero como, otra vez, la forma funcional no tiene grados mayores a uno entonces estos operadores no afectan la fórmula.

M_v pregunta por la siguiente fórmula:

$$A_{\mathcal{F}} = \coprod_{z_3} Rz_1 Rz_2 Rz_3[8 - 6z_3]$$

M_p proporciona el valor de $-6 = A_{\mathcal{F}} \rightarrow r_0$ y $p_1(z_3) = 8 - 6z_3$. M_v checa $p_1(0) + p_1(1) - p_1(0)p_1(1) = -6$ y genera $\alpha = 3$. M_p proporciona $p_2(z_3) = 8 - 6z_3$, M_v checa que $p_2(z_3)\, mod\, (z_3^2 - z_3) \mid_3 = -10 = p_1(3)$, genera $\beta = 2$ y calcula $p_2(\beta) = -4$. Como se ha llegado a que $A_{\mathcal{F}}$ es constante entonces aceptamos el valor inicial proporcionado por M_p, es decir, $\mathcal{F} = true$.

Este ejemplo nos muestra como funciona operativemente el Sistema Interactivo de Prueba definido. Si observamos su funcionamiento, en caso de que en algún momento M_v no pudiera verificar alguna de las condiciones, entonces M_v rechazará la palabra dada como entrada.



Capítulo 6

Conclusiones.

El Teorema **IP = PESPACIO** ha tenido gran relevancia en Teoría de Computación. Las técnicas usadas en la prueba: aritmetización de fórmulas booleanas y reducción de grado en polinomios, fueron usadas después en otras investigaciones para probar otros resultados similares (por ejemplo, en [3] se prueba que **NEXPTIME = MIP**, donde **MIP** es la clase de lenguajes que son reconocidos por Sistemas Interactivos de Prueba con Multi-Probadores, se utilizan estas técnicas).

La prueba del Teorema incitó una serie de resultados, los cuales llevaron a que en [16] se llegara a demostrar que **NP = PCP**($\log n$,1), donde **PCP**(f,g) representa la clase de todos los lenguajes con Pruebas Verificables Probabilísticamente que usan $O(f)$ bits aleatorios y observan $O(g)$ bits de prueba. Estos resultados tienen importancia primordialmente teórica, pues nos proporcionan otras caracterizaciones de **NP** (la implementación de estas técnicas es en la mayoría de los casos bastante difícil de crear) que podrían ayudar a aclarar las relaciones entre **NP** y **EXPTIEMPO** [11].

El Teorema **IP = PESPACIO** también ha tenido consecuencias importantes en el área de *Algoritmos de Aproximación*. Muchos problemas de Optimización Combinatoria (ej. Cubierta de Vértices, Ciclo Hamiltoniano en una Gráfica, Clán máximo en una Gráfica, etc) de gran importancia práctica, se ha demostrado que pertenecen a **NP**-completo. Dadas las grandes implicaciones que tiene encontrar algoritmos óptimos para estos problemas, muchos investigadores trabajaron para encontrar algoritmos de tiempo polinomial que generaran soluciones "aproximadas a las óptimas" a esta clase de problemas.

En [20] se prueba que cualquier algoritmo de aproximación suficiente-

mente bueno para el problema de Clán máximo en una Gráfica podría ser usado para probar que una Prueba Verificable Probabilísticamente existía y entonces determinar la membresía del algoritmo en **NP**-completo. Es decir, se prueba que encontrar buenas soluciones aproximadas a el problema del Clán Máximo es igual de difícil que encontrar soluciones exactas al mismo problema (pués el problema del Clán Máximo es **NP**-completo).

Bibliografía

[1] L. Babai. Trading group theory for randomness. *ACM 17th Symposium on Theory of Computing*, pages 421–429, 1985.

[2] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hirierarchy of complexity calses. *Journal of Computing*, (36):254–276, 1988.

[3] L. Babai and S. Moran. Non-deterministic exponential time has two-prover interactive protocol. In *Proceedings 31st. Ann. Symp. on Foundations of Computer Science*, Los Angeles, 1990. IEEE, IEEE Computer Science. Journal version (same title): *Computational Complexity* **1**.

[4] Richard Beigel. Class notes on interactive proof system. Calss notes, Dept. of Computer Science, Yale University, Dept. of Computer Science, Yale, January 1993.

[5] Papadimitriou C. Games against nature. *Proc. 24th Ann. IEEE Symposium Foudations of Computer Science*, pages 446–450, 1983.

[6] P. Feldman. The optimum prover lives in pspace. *Unpublished manuscript.*

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability.* Addison-Wesley, Reading, Massachusetts, first edition, 1979.

[8] Eiton M. Gurari. *An Introduction to Theory of Computation.* Computer Science Press, N.Y., New York, 1989.

[9] G. H. Hardy and E. M. Wright. *An introduction to the Theory of Numbers.* Oxford Science Publicatins, fifth edition, 1979.

[10] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languajes, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[11] David S. Johnson. The np-completness column:an ongoing guide. *Journal of Algorithms*, 13:502–524, 1992.

[12] Josep Díaz José Luis Balcazar and Joaquim Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Teoretical Computer Science.* Springer-Verlag, Berlin, Heidelberg, first edition, 1988.

[13] Josep Díaz José Luis Balcazar and Joaquim Gabarró. *Structural Complexity II*, volume 22 of *EATCS Monographs on Teoretical Computer Science.* Springer-Verlag, Berlin, Heidelberg, first edition, 1990.

[14] A. Widgerson O. Goldreich, S. Micaly. Proofs that yield nothing but their validity or all languajes in np have zero-knowledge. *Journal of Asociation for Computing Machinery*, 38(1):691–729, July 1991.

[15] M. Sipser O. Goldreich, Y. Mansur and S. Zachos. On completness and soundness in interactive proof systems. *Journal of Computing*, 18(1):429–442, February 1989.

[16] R. Motwani M. Sudan S. Arora, C. Lund and M. Szegedy. Proof verification and intractability of approximation problems. *Unpublished manuscript (April 1992).*

[17] S. Micaly S. Goldwasser and C. Rackoff. The knowledge complwxity of interactive proof-systems. *Journal of Computing*, (18):186–208, 1989.

[18] Adi Shamir. Ip = pspace. *Journal of the Asociation for Computing Machinery*, 39(4):869–877, October 1992.

[19] A. Shen. Ip = pspace : Simplified proof. *Journal of the Asociation for Computing Machinery*, 39(4):878–880, October 1992.

[20] L. Lováz U. Feige, S. Goldwasser and M. Szegedy. Approximation clique is almost np-complete. In *Proceedings 32nd. Ann. ACM. Symp. on Theory of Computing*, Los Angeles, 1991. IEEE, IEEE Computer Science.