Capítulo 1

La Jerarquía Polinomial

1.1. Introducción

Hay problemas que no se logran clasificar correctamente en las clases **P** y **NP** ya que no se conocen algoritmos de tiempo polinomial para ellos, o bien un testigo para dicho problema es difícil de encontrar (o bien, simplemente no existe). Una primera intención consistiría en clasificar dichos problemas en la clase **PSPACE**, pero surge la pregunta si es que no hay una categorización más fina para dichos problemas. Por ejemplo, consideremos el siguiente problema: "Sea C un circuito booleano, ¿es cierto que no hay un circuito con menos entradas tal que compute la misma función booleana?".

Para este problema, podríamos pensar en un testigo. Este podría ser un circuito de tamaño menor, lo que tendría tamaño polinomial con respecto al número de compuertas lógicas del circuito dado. Pero chequear que computa la misma función claramente no lleva tiempo polinomial, puesto que hay que chequear para todas las posibles combinaciones de la entrada del circuito. Claramente el problema se encuentra en **PSPACE** puesto que no es necesario más que una cantidad de espacio polinomial para chequear cada combinación de valores de la entrada, y el circuito también es de tamaño polinomial. La pregunta es, nuevamente, determinar si podemos refinar la clasificación del problema entre las clases **NP** y **PSPACE**.

1.2. Interludio de lógica de Predicados de Primer Orden

Es bueno notar una diferencia en la formulación del problema anterior con respecto a un problema **NP**. Consideremos por ejemplo el probema del circuito hamiltoniano: "Dado un grafo G, ¿existe un ciclo que cubra a todos los vértices exactamente una vez?". Se podría pensar en codificarlo de la siguiente manera: "Existe un testigo x que satisfaga la relación $R^1(\cdot)$". Por otra parte, el problema del circuito de la introducción se codifica como "Para todo testigo x, existe (otro) testigo y tal que satisfaga la relación $R^2(\cdot)$".

Aquí se emplea la notación R^n para denotar una relación n-aria.

Es fácil codificar estos problemas empleando el lenguaje de la lógica de primer orden. Este es un conjunto de símbolos, los llamados lógicos, que son los de puntuación (paréntesis), los conectivos lógicos ($\neg$, $\wedge$, $\vee$, $\rightarrow$), el símbolo de la igualdad, cuantificador universal ($\forall$) y existencial ($\exists$) y un conjunto numerable de variables (en la literatura tradicional a veces se definen sólo los símbolos $\neg$, $\rightarrow$, $\forall$ y la igualdad, dado que los demás pueden escribirse en función de estos cuatro). También existen los símbolos no lógicos, que corresponden a símbolos constantes (usualmente se escriben en minúsculas), símbolos de operaciones de aridad fija (f^j, g^k, j y k-arias, respectivamente) y relaciones de aridad fija (R^n, S^m, de aridad n y m, respectivamente).

Un término t del lenguaje puede ser un símbolo constante, un símbolo de variable, o si f^n es un síı mbolo de función n-aria, $f(t_1, t_2, t_3, \ldots, t_n)$ también es un término.

Una fórmula puede se construida de la siguiente manera: t_1, t_2, ..., t_n son términos, $=$ es el símbolo de igualdad, $t_i = t_j (i, j \leq n)$ es fórmula, si R^n es símbolo de relación, $R(t_1, \ldots, t_n)$ es fórmula, si φ y ψ son fórmulas, $\neg\varphi$, y $\varphi \star \psi$ son fórmulas, si es que $\star$ corresponde a los conectivos lógicos binarios y finalmente, si φ es fórmula, $\forall\varphi$ y $\exists\varphi$ también lo son.

Se pueden construir fórmulas para representar conocimiento. En particular, nos va a interesar las relaciones entre los cuantificadores. Una fórmula es verdadera cuando en algún contexto la fórmula representa algo verdadero en el mismo. En este sentido, la fórmula $\forall x \forall y (R(x, y) \rightarrow \forall z (R(y, z) \rightarrow R(x, z)))$ será verdadera en todo contexto en que la relación R^2 representa a una relación transitiva.

Con estas definiciones, podemos codificar el problema del circuito hamiltoniano con una fórmula de la siguiente forma:

$$\exists y R(x)$$

Donde y es un testigo para una instancia SI, R representa las condiciones para que sea un circuito hamiltoniano. No nos preocuparemos de cómo se define R, sino de la cuantificación del problema. Si $R(x)$ es verificable en tiempo polinomial, correspondería a la definición de un problema **NP**.

Por otra parte se tiene que el problema del circuito se puede codificar con algo del siguiente estilo:

$$\forall x \exists y R(x, y)$$

En este caso x representa una entrada del circuito e y representa el circuito reducido. R^2 representa la condición de que para cada entrada, el circuito debe tener la misma salida. Notar que la dificultad para encontrar un testigo consiste en que hay que chequear todos los elementos de un dominio, lo que, en este caso hace imposible encontrar testigos adecuados. Si tuviésemos problemas del tipo

$$\exists x_1 \exists x_2 \ldots \exists x_n R(x_1, \ldots, x_n)$$

El problema no parece tan difícil. Basta con encontrar un testigo para una cantidad fija de subproblemas. Pero si alternamos los cuantificadores, el problema consiste en encontrar testigos para todo un conjunto de otros posibles, o bien, si es que tenemos una formulación del tipo: $\exists x \forall y R(x, y)$, entonces el problema consiste en encontrar para algún testigo x, todos los testigos y, lo que naturalmente no es fácil.

1.3. Las Clases Σ y Π

Con lo anterior estamos en condiciones de dar una definición precisa para las clases en que se encuentran problemas como el del circuito booleano descrito anteriormente. Notemos que si bien, este problema podría ser resuelto en tiempo exponencial, nos interesa saber si es que dicho problema, dada su particular formulación lógica, posee alguna otra propiedad intersante, en particular, nos interesa saber si es que eiste una cota infrerior para el tiempo necesario para resolverlo, mejor que la exponencial. Para eso necesitamos dar algunas definiciones previas.

Definición 1.1: $\mathfrak{L} \in \Sigma_1^p$ si y sólo si existe una relación binaria, acotada polinomialmente ($|y| \leq p(|x|)$), calculable en tiempo polinomial, tal que:

$$x \in \mathfrak{L} \Leftrightarrow \exists y R(x, y)$$

Es fácil ver que de la formulación de esta clase nos estamos refiriendo a la clase **NP**, ya conocida. Sabemos que un problema están en **NP** si es que existe un testigo de tamaño polinomial para una instancia SI. Notar que el hecho que la relación sea "acotada"polinomialmente indica que el largo del testigo debe ser polinomial, mientras que al decir que es calculable en tiempo polinomial, podemos asimilarlo a la verificación del testigo.

Como el lector ya debe haber sospechado, existe una formulación de este tipo para los problemas de la clase **co-NP**, la cual, es simétrica a la anterior.

Definición 1.2: $\mathfrak{L} \in \Pi_1^p$ si y sólo si existe una relación binaria, acotada polinomialmente ($|y| \leq p(|x|)$), calculable en tiempo polinomial, tal que:

$$x \in \mathfrak{L} \Leftrightarrow \forall y R(x, y)$$

Estos problemas son bien conocidos ya. Nos interesa ahora abordar los problemas cuya formulación presenta alternación de cuantificadores. Podríamos considerar la idea de componer las clases Σ_1^p y Pi_1^p, de modo de representar estos problemas y aprovechar de dar una semántica adecuada, así como una clasificación útil.

Volviendo a nuestro ejemplo del circuito booleano, podríamos pensar que corresponde a un problema con una típica codificación **co-NP** sobre un problema codificado como un problema **NP**. Asimismo hay muchos problemas que se pueden codificar de esta forma, y que se pueden ir "componiendo" de manera de obtener problemas cada vez más complejos. Esto motiva las siguientes definiciones.

Definición 1.3: Para $k \geq 1$ $\mathfrak{L} \in \Sigma_k^p$ si y sólo si existe una relación de aridad $k+1$ $R(x, y_1, \ldots, y_n)$, tal que es acotada polinomialmente $(\sum_i |y| \leq p(|x|))$ y calculable en tiempo polinomial tal que:

$$x \in \mathfrak{L} \Leftrightarrow \exists y_1 \forall y_2 \exists y_3 \ldots R(x, y_1, \ldots, y_k)$$

Nuevamente, tendremos la definición simétrica para la clase Π_k^p, al igual que en el caso anterior.

Definición 1.4: Para $k \geq 1$ $\mathfrak{L} \in \Pi_k^p$ si y sólo si existe una relación de aridad $k+1$ $R(x, y_1, \ldots, y_n)$, tal que es acotada polinomialmente $(\sum_i |y| \leq p(|x|))$ y calculable en tiempo polinomial tal que:

$$x \in \mathfrak{L} \Leftrightarrow \forall y_1 \exists y_2 \forall y_3 \ldots R(x, y_1, \ldots, y_k)$$

Notar que llamamos Σ_k^p a la clase de problemas en cuyo planteamiento tiene $k-1$ alternación de cuantificadores, cuando el primero es existencial. La única diferencia con Π_k^p es que el primer cuantificador es universal. En este sentido, un problema en Σ_k^p consistiría en encontrar (al menos) un testigo, tal que satisfaga un problema en Pi_{k-1}^p. Simétricamente, un problema en Pi_k^p consiste en chequear que todos los posibles testigos satisfagan un problema en Σ_{k-1}^p.

Podemos notar en el caso de nuestro ejemplo del circuito booleano que, con respecto al circuito original, un circuito reducido tiene un número polinomial de entradas, y el chequeo de una entrada también es polinomial. Este problema, como ya debe suponer el lector, corresponde a uno en la clase Σ_2^p. Por último se define la "jerarquía polinomial acumulativa" como la clase **PH**, donde

$$\mathbf{PH} = \bigcup_{i \geq 1} \Sigma_i^p$$

En la siguiente sección analizaremos los oráculos, que son herramientas que nos permitirán analizar estos problemas desde una perspectiva sencilla, y mediante los cuales es fácil analizar relaciones de inclusión de estos problemas.

1.4. Oráculos

Antes de dar la definición formal de oráculo, se pueden dar algunas ideas de lo que es y del poder de análisis que provee.

Un Oráculo puede verse como un supercomputador que responde consultas en tiempo $O(1)$. Como el lector ya debe haber advertido, su utilidad es puramente analítica, debido a que es imposible realizar esta operación para un problema arbitrario.

Empleando un oráculo es posible analizar un problema haciendo abstracción de ciertos subproblemas. En este sentido, es fácil ver que la complejidad del problema completo corresponde a la complejidad del problema inicial, compuesta de alguna forma con la complejidad del subproblema.

Una de las principales ventajas del uso de oráculos es que permite identificar fácilmente la clase de complejidad del problema, dado que en términos de análisis basta con identificar un oráculo adecuado y la clase de complejidad del problema "principal".

En el caso de la jerarquía polinomial, es evidente su aplicación. Por ejemplo, haciendo una simplificación de lo que se ha expuesto, un problema de la clase Σ_k^p puede verse como un problema Π_{k-1}^p con un cuantificador existencial que lo precede. eventualmente podríamos analizar este problema como un problema **NP**, y hacemos una consulta a un oráculo para Π_{k-1}^p. Este tipo de análisis es el que deseamos poder lograr, pero, para alcanzar este objetivo, es necesario formalizar la definición de un oráculo.

Definición 1.5: Una *máquina de Turing con oráculo* es una máquina de Turing determinística con múltiples cintas, una de las cuales es llamada "cinta de consulta", y tres estados especiales: el estado de consulta q_C, y los estados de respuesta q_{SI} y q_{NO}.

Sea $A \subseteq \Sigma^*$ un lenguaje cualquiera. Una máquina de Turing M con oráculo A funciona del mismo modo que una máquina de Turing tradicional, excepto por las transiciones del estado de consulta: desde él, la máquina M se mueve al estado q_{SI} si el string que se encuentra en la cinta de consulta pertenece al lenguaje A, o al estado q_{NO} en caso contrario.

La computación de M con oráculo A sobre una entrada x se escribe como $M^A(x)$.

La complejidad de tiempo de las máquinas de Turing con oráculos se define del mismo modo que para las máquinas de Turing tradicionales: cada consulta al oráculo, es decir, cada paso del estado de consulta a uno de los estados de respuesta, cuenta como cualquier otro paso. Esto equivale a suponer que la evaluación de la pertenencia del string contenido en la cinta de consulta al lenguaje del oráculo se realiza en tiempo constante. Sin embargo, es necesario considerar el tiempo que toma escribir el string en la cinta de consulta.

También es posible definir máquinas de Turing no determinísticas con oráculos del mismo modo en que se ha definido a las determinísticas. Luego, si $\mathcal{C}$ es una clase de complejidad de tiempo cualquiera, ya sea determinística o no, es posible definir $\mathcal{C}^A$ como la clase de todos los lenguajes decididos (o aceptados) por máquinas del mismo tipo y tiempo acotado del mismo modo que en $\mathcal{C}$, sólo que las máquinas ahora tienen oráculo A. Dicho de otra forma, es posible resolver el problema, empleando los recursos necesarios para la clase $\mathcal{C}$, consultando a un oráculo A.

Es importante notar que no se considera el tiempo que el oráculo se demora en contestar, por lo que asumimos que es $O(1)$. En este sentido, también podemos observar que un oráculo determinístico siempre responde SI o NO. De este modo, un oráculo para un lenguaje $\mathcal{C}$, también lo es para su complemento. Son muchos los resultados que se pueden obtener empleando oráculos, aquí se muestran sólo algunos.

Teorema 1.1: Existe oráculo A tal que $\mathbf{P}^A = \mathbf{NP}^A$.

Demostración: Sea A cualquier lenguaje **PSPACE**-completo. Se tiene que

$$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{NPSPACE} \subseteq \mathbf{PSPACE}$$

Luego, $\mathbf{P}^A = \mathbf{NP}^A$.

La primera inclusión se cumple debido a que A es **PSPACE**-completo. Luego, cualquier lenguaje $L \in \mathbf{PSPACE}$ puede ser decidido por una máquina de Turing determinística de tiempo polinomial, que realiza una reducción de L a A, y luego usa el oráculo una vez.

La segunda inclusión es trivial.

Para la tercera inclusión, cualquier máquina de Turing no determinística de tiempo polinomial con oráculo A puede ser simulada por una máquina de Turing no determinística de espacio acotado polinomialmente, que resuelve las consultas a A por sí misma, en espacio polinomial.

La última inclusión corresponde al teorema de Savitch. $\square$

Teorema 1.2: Existe oráculo B tal que $\mathbf{P}^B \neq \mathbf{NP}^B$.

Demostración: Es necesario encontrar un lenguaje $L \in \mathbf{NP}^B - \mathbf{P}^B$.

Sea L el siguiente lenguaje unario:

$$L = \{0^n : \text{existe un } x \in B \text{ con } |x| = n\}$$

Es claro que $L \in \mathbf{NP}^B$: una máquina de Turing no determinística con oráculo B podría adivinar un x de largo n al procesar la entrada 0^n, y usar el oráculo para verificar que $x \in B$.

Ahora es necesario definir $B \subseteq \{0,1\}^*$ de modo tal que $L \notin \mathbf{P}^B$. Suponga que tenemos una enumeración $M_1^?, M_2^?, \ldots$ de todas las máquinas de Turing determinísticas de tiempo polinomial con oráculo. Asumimos que cada máquina aparece en esta enumeración *infinitas veces*. Cualquier enumeración satisface esta condición, pues cualquier máquina puede ser modificada de manera arbitraria agregando estados inútiles que no afectan al lenguaje que se decide, y todas estas variantes deben aparecer en algún lugar de la enumeración.

Definimos B en etapas. Al principio de la etapa i ya hemos calculado B_{i-1}, el conjunto de todos los strings en B de largo menor que i. También tenemos un conjunto $X \subseteq \Sigma^*$ de *excepciones*, strings que debemos recordar no poner en B. Inicialmente, al comienzo de la primera etapa, $B_0 = X = \emptyset$.

Para definir B_i, es decir, para decidir qué strings de largo i debieran pertenecer a B, simulamos $M_i^B(0^i)$ por $i^{\log i}$ pasos. Note el número de pasos: ha sido escogida una función mucho menor que exponencial, y aún asintóticamente mayor que cualquier polinomial.

Durante la simulación de $M_i^B(0^i)$, la máquina con oráculo puede realizar una pregunta de la forma "¿$x \in B$?". Para responder la pregunta, simplemente buscamos x en B_{i-1} si $|x| < i$. Si se encuentra en B_{i-1}, respondemos SI, es decir, la máquina M_i^B se mueve al estado q_{SI}, o a q_{NO} si x no se encuentra en B_{i-1}. Pero si $|x| \geq i$ la máquina se va al estado q_{NO} – respondemos "NO" – y agregamos x al conjunto X de excepciones: queremos recordar el compromiso que $x \notin B$.

Suponga que al final, luego de $i^{\log i}$ o menos pasos, la máquina rechaza. Recuerde que queremos impedir que M_i^B decida L. Definimos B_i como $B_{i-1} \cup \{x \in \{0,1\}^* : |x| = i, x \notin X\}$. De este modo nos aseguramos que $0^i \in L$ (dado que existe un string de largo i en B, recuerde la definición de L), y M_i^B, que ha rechazado 0^i, no puede ser la máquina que decide L: $L(M_i^B) \neq L$. Pero, ¿cómo sabemos que el conjunto $\{x \in \{0,1\}^* : |x| = i, x \notin X\}$ es no vacío? Lo sabemos debido a que X contiene no más de $\sum_{j=1}^{i} j^{\log i}$ strings de largo i (éste es el número total de pasos simulados hasta ese momento en todas las máquinas), y un poco de cálculo demuestra que esa suma es siempre menor que 2^i, el número de strings en $\{0,1\}^i$. Si M_i^B acepta 0^i en el tiempo especificado, entonces definimos $B_i = B_{i-1}$. Nuevamente, esto asegura que $0^i \notin L$, y por lo tanto, $L(M_i^B) \neq L$.

Pero es posible que $M_i^B(0^i)$ no se detenga luego de $i^{\log i}$ pasos. La cota polinomial $p(n)$ de la máquina puede ser muy grande, y el valor de i muy pequeño, tal que $i^{\log i} < p(i)$. Si esto ocurre, definimos $B_i = B_{i-1}$ como si la máquina hubiera aceptado. Desde luego, esto no asegura que $L(M_i^B) \neq L$. Sin embargo, máquinas equivalentes a ésta aparecen infinitamente a menudo en la enumeración, por lo que eventualmente aparecerá una máquina M_I^B donde el índice I es suficientemente grande tal que $I^{\log I} \geq p(I)$. Esto asegurará que $L \neq L(M_I^B) = L(M_i^B)$.

Continuando de este modo, definimos completamente el oráculo B. Dado que esta definición de B sistemáticamente evita que todas las máquinas de Turing de tiempo polinomial con oráculo B decidan L, podemos concluir que $L \notin \mathbf{P}^B$, y entonces la demostración queda completa. $\square$

Teorema 1.3: **P**=**NP** si y sólo si para todo oráculo x, $\mathbf{P}^x = \mathbf{NP}_p^x$ ($\mathbf{NP}_p^x$ denota que se puede emplear el oráculo x sólo un número polinomial de veces).

1.5. Jerarquía Polinomial con Oráculos

Se puede notar que, para que una Máquina de Turing pueda resolver un problema, digamos en Σ_2^p, tendría que resolver un problema **co-NP** y sobre ésta solución, resolver un problema **NP**. Podría simplemente resolver un problema **NP**, y consultar al oráculo por el problema **co-NP** asociado. En este sentido, podríamos pensar en que $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$ (recordar que on oráculo **NP** es equivalente a uno **co-NP**).

En este sentido, es fácil ver que la jerarquía polinomial se puede escribir en los siguientes términos:
$\Sigma_k^p = \mathbf{NP}^{\Sigma_{k-1}^p}$
$\Pi_k^p = \textbf{ co-NP }^{\Sigma_{k-1}^p}$

Veremos que ambas definiciones para la jerarquía polinomial son equivalentes.

Teorema 1.4: Sea L un lenguaje, y considere $i \geq 1$. $L \in \Sigma_i^p$ si y sólo si existe una relación R polinomialmente balanceada tal que el lenguaje $\{x; y : (x, y) \in R\}$ esté en Π_{i-1}^p y

$$L = \{x : \text{ existe } y \text{ tal que } (x, y) \in R\}$$

Demostración: Por inducción sobre i. Para $i = 1$ se tiene la definición de $\Sigma_1^p = \mathbf{NP}$ de acuerdo a lo visto al principio de la sección 1.3. Luego, considere $i > 1$ y que la relación R existe. Debemos demostrar que $L \in \Sigma_i^p$, es decir, debemos describir una máquina de tiempo polinomial no determinística, con un lenguaje en Σ_{i-1}^p como oráculo, que decida L. Tal máquina es fácil de construir: la máquina no determinística simplemente adivina en la entrada x un y apropiado, y pregunta a un oráculo Σ_{i-1}^p si $(x, y) \in R$ (más precisamente, dado que R es una relación en Π_{i-1}^p, es necesario consultar si $(x, y) \notin R$).

Ahora, suponga que $L \in \Sigma_i^p$. Debemos demostrar que existe una relación R apropiada. Lo que sabemos es que L puede ser decidido por una máquina de Turing $M^?$, no determinística de tiempo polinomial, con un lenguaje oráculo $K \in \Sigma_{i-1}^p$. Dado que $K \in \Sigma_{i-1}^p$, por inducción existe una relación S reconocible en Π_{i-2}^p tal que $z \in K$ si y sólo si existe un w tal que $(z, w) \in S$.

Debemos describir una relación R para L polinomialmente balanceada, y decidible en tiempo polinomial, es decir, un certificado para cada $x \in L$. Sabemos que $x \in L$ si y sólo si existe una computación correcta de M^K con entrada x y que termina en un estado de aceptación. El certificado de x será un string y que grabe tal computación de M^K. Pero recuerde que M^K es ahora una máquina con oráculo en $K \in \Sigma^p_{i-1}$. Luego, varios de sus pasos serán consultas al oráculo K. Algunos tendrán respuesta positiva y otros negativa. Por cada consulta positiva z_i, el certificado y también incluirá el certificado w_i de z_i tal que $(z_i, w_i) \in S$. Luego, la definición de R es la siguiente:

$(x, y) \in R$ si y sólo si y graba una computación de $M^?$ con entrada x que lo acepta, junto con un certificado w_i por cada consulta positiva z_i en la computación.

Revisar si $(x, y) \in R$ puede ser realizado en Π^p_{i-1}. Primero, debemos revisar si todos los pasos realizados por $M^?$ son legales. Ésto puede hacerse en tiempo determinístico polinomial. Luego debemos revisar, para un número polinomial de pares (z_i, w_i) si $(z_i, w_i) \in S$. Pero esto puede ser hecho en Π^p_{i-2}, y, por lo tanto, también en Π^p_{i-1}. Finalmente, para cada consulta negativa z'_i debemos revisar si realmente $z'_i \notin K$. Pero como $K \in \Sigma^p_{i-1}$, ésta es otra consulta Π^p_{i-1}. Luego, $(x, y) \in R$ si y sólo si varias consultas Π^p_{i-1} tienen respuesta positiva. Ésto puede hacerse en una computación en $\Pi_{i-1}p$. □

Del teorema anterior se desprende el siguiente resultado "dual":

Corolario: Sea L un lenguaje, y considere $i \geq 1$. $L \in \Pi^p_{i-1}$ si y sólo si existe una relación binaria R polinomialmente balanceada tal que el lenguaje $\{(x; y) : (x, y) \in R\}$ está en Σ^p_{i-1} y

$$L = \{x : \text{ para todo } y \text{ con } |y| \leq |x|^k, (x, y) \in R\}$$

***Demostración*:** Sólo recuerde que Π^p_i es precisamente co-Σ^p_i. □

Finalmente, tenemos el siguiente corolario:

Corolario: Sea L un lenguaje, y considere $i \geq 1$. $L \in \Sigma^p_i$ si y sólo si existe una relación R de aridad $i + 1$, polinomialmente balanceada, decidible en tiempo polinomial, tal que

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 \dots Q y_i \text{ tal que } (x, y_1, y_2, \dots, y_i) \in R\}$$

donde el iésimo cuantificador Q es "para todo" si i es par, y "existe" si es impar.

***Demostración*:** Reemplace repetidamente los lenguajes en Π^p_j o Σ^p_j por sus respectivos certificados, como se hizo en el teorema y corolario anteriores. □

Esta caracterización de la jerarquía polinomial nos permite ver, de manera bastante natural, varias características importantes.

Es evidente bajo esta perspectiva que si una Máquina de Turing puede resolver un problema **NP**, consultando a un oráculo de la clase Σ^p_k, entonces también puede resolverlo consultando a un oráculo Σ^p_{k+1}. En este sentido, un oráculo de una clase en una posición superior en la jerarquía, puede resolver problemas que se encuentran en lugares inferiores de la misma. Se puede observar entonces que para todo $k \geq 1$, $\Sigma^p_k \subseteq \Sigma^p_{k+1}$.

Asimismo, también se pueden observar las inclusiones $\Pi_k^p \subseteq \Pi_{k+1}^p$, $\Sigma_k^p \subseteq \Pi_{k+1}^p$ y $\Pi_k^p \subseteq \Sigma_{k+1}^p$, para $k \geq 1$.

En este punto se pueden ver varias de las caracterísicas de estas clases. Sólo mencionaremos algunas:

Teorema 1.5: Si para algún $k \geq 1$ $\Sigma_k^p = \Pi_k^p$, entonces para todo $j > k$ se tiene que $\Sigma_j^p = \Pi_k^p = \Sigma_k^p$.

Demostración: Por inducción sobre k. Si $i = k$, enonces, trivialmente $\Sigma_i^p = Pi_i^p$. Ahora, supongamos que $\Sigma_{i-1}^p = Pi_{i-1}^p$ y que $i > k$. Por definición, se tiene que $\mathfrak{L} \in \Sigma_i^p \Leftrightarrow$ existe $\mathfrak{L}' \in \Pi_{i-1}^p$ tal que $x \in \mathfrak{L} \Leftrightarrow \exists(x, y) \in \mathfrak{L}'$. Pero $\Sigma_{i-1}^p = \Pi_{i-1}^p = \Sigma - k^p = Pi_k^p$. Luego, $\mathfrak{L} \in \Sigma_{k+1}^p$ para todo $k < i$. Por hipótesis de inducción, $\mathfrak{L} \in Pi_{k+1}^p$. Por lo tanto, $\Sigma_i^p \subseteq \Sigma_{k+1}^p = \Sigma_k^p$. Finalmente: $\Sigma_i^p = \Sigma_k^p = \Pi_k^p$. □

Corolario: NP = **co-NP** entonces la jerarquía colapsa a su primer nivel. □

Existen problemas completos para Σ_k^p y Π_k^p. No está resuelto si el problema del circuito booleano descrito en la introducción es Π_2^pcompleto o no. Un ejemplo de problema completo para todo Σ_i^p es $QSAT_i$: "Dado una expresión booleana φ con variables booleanas particionadas en i conjuntos $X_1, \ldots X_i$, ¿es cierto que para toda asignación parcial de valores de verdad para las variables de X_1 hay una asignación parcial para las variables de X_2 tal que para toda asignación parcial de variables en X_3, hasta X_i, φ es satisfecha por toda la asignación?".

Teorema 1.6: Si existiese un problema **PH**-completo, la jerarquía polinomial colapsa sobre algún nivel k.

Supongamos que $\mathfrak{L}$ es **PH**-completo. Dado que $\mathfrak{L} \in$ **PH**, existe un $i > 1$ tal que $\mathfrak{L} \in \Sigma_i^p$. Pero cualquier lenguaje $\mathfrak{L}' \in \Sigma_i^p$ se reduce a $\mathfrak{L}$. Con esto, se tiene que $\mathfrak{L}' \in \Sigma_i^p$, luego $\Sigma_i^p = \Sigma_{i+1}^p$. □

Es fácil ver que para cualquier problema en Σ_k^p, éste se puede resolver empleando espacio polinomial. Entonces se tiene que **PH** $\subseteq$ **PSPACE**. Pero sigue estando abierta la pregunta ¿**PSPACE**=**PH**?.

1.6. Las clases Δ

1.6.1. La clase DP

Hasta ahora nos hemos referido sólo a los problemas que están en Σ_i^p y Π_i^p. Hemos tratado las inclusiones que hay entre estas clases y varias propiedades de éstos. Pero el lector probablemente se esté preguntando si es que las intersecciones de estas clases dan lugar a alguna clase con problemas interesantes. La respuesta es afirmativa, veremos la clase **DP** y las clases Δ, que tienen relación con este hecho.

Definición 1.6: Un lenguaje L se encuentra en la clase **DP** si y sólo si existen lenguajes $L_1 \in \mathbf{NP}$ y $L_2 \in \mathbf{co\text{-}NP}$ tales que $L = L_1 \cap L_2$.

Hay que advertir que **DP** no corresponde a **NP** $\cap$ **co-NP**. Hay evidencia que **DP** no está contenido en **NP** $\cup$ **co-NP**.

Se puede definir **DP** en términos de las llamadas que debe hacer una Máquina de Turing a oráculos para resolver los problemas en esta clase. La máquina realiza dos llamados a un oráculo SAT, y acepta el lenguaje si y sólo si es que la primera respuesta es SI y la segunda NO.

Se puede deducir, de la formulación de la clase **DP** que $\mathbf{DP} \subseteq \Sigma_2^p$ y $\mathbf{DP} \subseteq \Pi_2^p$.

Analicemos el problema SAT-UNSAT: "Dadas dos expresiones booleanas, φ y φ', ambas en Forma Normal Conjuntiva (FNC), con tres literales por cláusula. ¿Es cierto que φ es satisfacible y φ' no?

Notar que las fórmulas en FNC son de la forma $(x_1 \vee x_2 \vee \ldots) \wedge (x_3 \vee x_4 \vee \ldots) \ldots$.

Teorema 1.7: SAT-UNSAT es **DP**-completo

***Demostración*:** Es necesario mostrar dos lenguajes: $\mathfrak{L}_1 \in \mathbf{NP}$ y $\mathfrak{L}_2 \in \mathbf{co\text{-}NP}$, tal que para el conjunto de todas las instancias SI de SAT-UNSAT sean $\mathfrak{L}_1 \cap \mathfrak{L}_2$. Consideremos $\mathfrak{L}_1 = \{(\phi, \phi')|\phi \text{ es satisfactible}\}$ y $\mathfrak{L}_1 = \{(\phi, \phi')|\phi' \text{ es insatisfactible}\}$.

Para demostrar completitud, tomemos $\mathfrak{L}$ cualquier lenguaje en **DP**. Hay que demostrar que $\mathfrak{L}$ se puede reducir a SAT-UNSAT, pero lo único que sabemos es que existen dos lenguajes $\mathfrak{L}_1 \in \mathbf{NP}$ y $\mathfrak{L}_2 \in \mathbf{co\text{-}NP}$, tales que $\mathfrak{L} = \mathfrak{L}_1 \cap \mathfrak{L}_2$. Dado que SAT es **NP**-completo, sabemos que hay una reducción R_1 de $\mathfrak{L}_1$ a SAT, y una reducción R_2 de $\mathfrak{L}_2$ a SAT. La reducción de L a SAT-UNSAT es la siguiente:

$$R(x) = (R_1(x), R_2(x))$$

Se tiene que R(x) es una instancia SI de SAT-UNSAT si y sólo si $R_1(x)$ es satisfacible y $R_2(x)$ no lo es. Todo esto es verdadero si y sólo si $x \in \mathfrak{L}_1$ y $x \in \mathfrak{L}_2$ o, equivalentemente $x \in \mathfrak{L}_1$. □

Otro problema interesante es MAX-Clique: "Sea $G = (V, E)$ un grafo, y $k \geq 1$, Verificar que el Clique más grande tiene tamaño k". Este problema puede resolverse naturalmente empleando dos llamadas a un oráculo **NP**. En la primera, le consultamos si es que el grafo tiene un clique de tamaño k. En la segunda llamada, preguntamos si es ue tiene un Clique de tamaño $k+1$. Notar que si un grafo no tiene un Clique de tamaño $k+1$, tampoco puede tener uno de tamaño $k+2$ ni mayor. En este caso, una instancia SI del problema es aquella en que la respuesta de la primera llamada al oráculo da como respuesta SI y la segunda NO.

Se pueden generalizar las clases de problemas que realizan llamadas a oráculos **NP** a un número fijo de llamadas a dichos oráculos, teniendo que la respuesta de la máquina va a depende de cierta fórmula booleana donde las variables son las respuestas del oráculo.

Una clase de problemas interesante con respecto a las llamadas a SAT, es aquella que se realiza un número *polinomial* de llamadas a SAT. Se denota con $\mathbf{P}^{\mathbf{NP}}$ (Corresponde a lo que denominaremos Δ_2).

1.6.2. Las clases Δ

Se define la clase $\Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$. Equivalentemente corresponde a los problemas que son solubles en tiempo polinomial con llamadas a oráculos Σ_i^p. En este sentido, se tiene que $\Delta_0 = \Delta_1 = \mathbf{P}^{\Sigma_0^p} = \mathbf{P}^{\mathbf{P}} = \mathbf{P}$.

De la formulación misma del problema, vemos que se dan las siguientes inclusiones:

$\Sigma_i^p \subseteq \Delta_{i+1}$
$\Pi_i^p \subseteq \Delta_{i+1}$
$\Delta_i \subseteq \Sigma_{i+1}^p$
$\Delta_i \subseteq \Pi_{i+1}^p$

En particular se puede observar que $\Delta_i \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$, pero, al igual que con la clase **DP**, hay que hacer la salvedad que Δ_{k+1} no es equivalente a $\Sigma_k^p \cap \Pi_k^p$.

Se puede demostrar también que **DP** $\subseteq \Delta_2$. esto es intuitivo, si es que observamos las definiciones de las clases con oráculos. **DP** es la clase que realiza dos llamados a un oráculo **NP**, mientras que Δ_2, realiza un número polinomial de llamados. Naturalmente, cualquier problema que se puede resolver con una Máquina de Turing llamando dos veces a un oráculo **NP**, se puede resolver también con una Máquina de Turing que realiza un número polinomial de llamados a oráculo **NP**. Este hecho representa una refinación mayor de la clasificación de los problemas que se encuentran en la jerarquía polinomial

1.7. Ejercios Propuestos

1. Verdadero o Falso:

   a) Si $\mathfrak{L}$ es **NP**-completo y $\mathfrak{L}'$ es **co-NP**-completo entonces $\mathfrak{L} \cap \mathfrak{L}'$ es **DP**-completo.

   b) Si $\mathfrak{L}$ es **NP**-completo, entonces $\mathfrak{L} \cap \mathfrak{L}^C$ es **DP**-completo.

2. Muestre que si **NP** $\subseteq$ **TIME**$(n^{\log n})$, entonces **PH**$\subseteq$**TIME**$(n^{\log^k n})$.

3. Muestre que **BPP**$\subseteq \Sigma_2^p$ (**BPP** corresponde a la clase de problemas que son solubles empleando algoritmos "Montecarlo").

4. Muestre que **BPP**$\subseteq \Sigma_2^p \cap \Pi_2^p$.

5. Muestre un oráculo A tal que **PH**$\neq$**PSPACE**.

6. Muestre un oráculo B tal que **PH** colapsa a un nivel arbitrario.

7. Explique por qué un algoritmo Min-Max que encuentre una jugada ganadora en k jugadas estaría en Σ_k^p.

8. Muestre que **PH**$\subseteq$**PSPACE** (HINT: Puede basarse en la idea de la pregunta anterior).

9. Muestre que $\Delta_k \subsetneq \Sigma_k^p \cap \Pi_k^p$