UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
POSTGRADO EN MATEMÁTICA

COMPLETITUD EN EL SEGUNDO NIVEL DE LA JERARQUÍA POLINOMIAL A TRAVÉS DE PROPIEDADES SINTÁCTICAS

Autor: Edwin Pin.
Tutor: Nerio Borges.

Trabajo de Grado de Maestría presentado ante la ilustre Universidad Central de Venezuela para optar al título de Magister Scientiarium, Mención Matemática.

Caracas, Venezuela
Julio 2016

Resumen

La Teoría de Complejidad Descriptiva se basa en la caracterización de clases de complejidad, problemas de decisión, reducciones y demás conceptos computacionales a través de sistemas lingüísticos formales como la lógica de primer orden, la lógica de segundo orden, entre otros. El resultado que dio inicio a este tipo de investigaciones se conoce como Teorema de Fagin y plantea que la clase de complejidad **NP** es el conjunto de problemas expresables mediante fórmulas de segundo orden existenciales. Stockmeyer generalizó este resultado al definir inductivamente nuevas clases de complejidad, denotadas por Σ_k^p, capturadas por fragmentos más generales de fórmulas de segundo orden y cuya unión da como resultado la clase de todos los problemas expresables en segundo orden. Esta última clase, denominada Jerarquía Polinomial, ha sido bastante estudiada desde su formulación. En este trabajo nos concentramos en el segundo nivel de la Jerarquía Polinomial, la clase de complejidad Σ_2^p.

Uno de los puntos más importantes que se estudian dada una clase de complejidad **C** es la propiedad de completitud. Un problema A se dice que es **C**-completo si cualquier otro problema en **C** se puede interpretar y resolver eficientemente a través de A. Desde que Stephen Cook demostrara en 1971 la existencia de problemas **NP**-completos, la cantidad de resultados obtenidos bajo este concepto han sido abrumadores. Desde la perspectiva de la Complejidad Descriptiva, un problema B se puede reducir a un problema A eficientemente, si existe una interpretación de B en A descrita mediante fórmulas de primer orden. En este ámbito, Immerman y Landau demostraron varias propiedades de un tipo especial de reducciones, denominadas como proyecciones. Si bien las proyecciones caracterizan la completitud en un sentido más restrictivo que cualquier otro tipo de reducción, la mayoría de los problemas naturales que se han demostrado completos en sus respectivas clases de complejidad también se muestran completos bajo este tipo de reducciones. Medina e Immerman demostraron que la clase de problemas **NP**-completos vía proyecciones también se puede caracterizar sintácticamente, en el mismo sentido que establece el Teorema de Fagin, es decir, existe un fragmento de fórmulas de segundo orden que caracteriza a la clase de problemas

NP-completos vía proyecciones. Este resultado fue generalizado por Borges y Bonet en el año 2008 para cualquier clase de complejidad **C** con ciertas características especiales. En ese mismo trabajo proporcionarían métodos de carácter sintáctico y combinatorio para demostrar la completitud de una extensa lista de problemas. Uno de tales métodos se denominó superfluidad.

La técnica de superfluidad se basa en el estudio de conjunciones de la forma $(\varphi \wedge \Phi)$, donde φ es una sentencia universal de primer orden y Φ es una fórmula sobre una lógica $\mathcal{L}$ que captura a una clase de complejidad **C**. Si $\mathcal{L}$ y **C** satisfacen ciertas propiedades, entonces la **C**-completitud del problema asociado a la fórmula $(\varphi \wedge \Phi)$ implica la **C**-completitud del problema asociado a Φ. Esta propiedad se estudió y se demostró válida por primera vez en las clases de complejidad **NL**, **P**, **NP** y **coNP**.

Siguiendo esa línea de investigación, en esta monografía se realiza un estudio exhaustivo de la clase de complejidad Σ_2^p, con el propósito de obtener la propiedad de superfluidad, la cual se ha probado válida tanto en esta clase como en su complemento. Se proporciona además una lista de problemas clasificados como Σ_2^p-completos vía proyecciones.

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
COMISIÓN DE ESTUDIOS DE POSTGRADO

Comisión de Estudios de Postgrado

VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de la Facultad de Ciencias de la Universidad Central de Venezuela, para examinar el **Trabajo de Grado de Maestría** presentado por el **LIC. EDWIN PIN BAQUE, CI. No. V-18.813.365**, bajo el título ***"COMPLETITUD EN EL SEGUNDO NIVEL DE LA JERARQUIA POLINOMIAL A TRAVES DE PROPIEDADES SINTACTICAS"***, a fin de cumplir con el requisito legal para optar al grado académico de **MAGISTER EN CIENCIAS, MENCIÓN MATEMÁTICA**, dejan constancia de lo siguiente:

1.- Leído como fue dicho trabajo por cada uno de los miembros del jurado, se fijó el día **03** de **Marzo** de **2017** a las **10:00 am**, para que el autor lo defendiera en forma pública, lo que éste hizo en la **Sala de Leandro Aristeguieta de la Facultad de Ciencias, Universidad Central de Venezuela**, mediante un resumen oral de su contenido, luego de lo cual respondió satisfactoriamente a las preguntas que le fueron formuladas por el jurado, todo ello conforme con lo dispuesto en el Reglamento de Estudios de Postgrado.

2.- Finalizada la defensa del trabajo, el jurado decidió **aprobarlo**, por considerar, sin hacerse solidario con la ideas expuestas por el autor, que se ajusta a lo dispuesto y exigido en el Reglamento de Estudios de Postgrado.

Para dar este veredicto, el jurado estimó que el trabajo examinado demuestra dominio completo del tema, la versión escrita está muy bien elaborada, trabajó de manera muy independiente, durante la presentación fue claro y respondió muy bien todas la preguntas del jurado.

3.- El jurado por unanimidad decidió otorgar la calificación de EXCELENTE al presente trabajo por considerarlo de excepcional calidad, ya que contiene aportes originales al área que pueden ser publicados en revistas internacionales de investigación.

En fe de lo cual se levanta la presente ACTA, a los **03** días del mes de **Marzo** del año **2017**, conforme a lo dispuesto en el Reglamento de Estudios de Postgrado, actuó como Coordinador del jurado **DR. NERIO BORGES (USB).**

DR. BLAI BONET (USB)
C.I.V-10.337.671
Jurado designado por el Consejo
de la Facultad

MSc. FEDERICO FLAVIANI (USB)
C.I.V-15.665.076
Jurado designado por el Consejo
de la Facultad

DR. NERIO BORGES (USB)
C.I.V-6.746.969
Tutor

POSTGRADO DE MATEMÁTICA

MM/mgf.-

Índice general

Introducción

Uno de los conceptos de mayor relevancia en Teoría de Complejidad Computacional es el de *completitud*. La formulación de este concepto, derivado de un resultado de Stephen Cook [8], condujo al planteamiento del famoso problema **P versus NP**.

Cook demostró que cualquier problema de **NP** puede ser reinterpretado eficientemente a través del Problema de Satisfabilidad Booleana, o SAT para recortar. SAT es sólo el primero de una extensa lista de problemas que se clasifican habitualmente como **NP**-*completos*. La importancia de esta clasificación radica en que el aparente contraste entre **P** y **NP** se debe principalmente a este tipo de problemas. La guía de problemas **NP**-completos de Garey y Johnson [10], publicada apenas 8 años después del trabajo de Cook, muestra lo rápido que fue acogido el concepto de completitud en el estudio formal de la computabilidad.

Otro resultado destacado, debido a Ron Fagin [9], establece que la clase de complejidad **NP** puede estudiarse mediante recursos expresivos de la lógica de segundo orden. Específicamente, el conjunto $\exists\mathbf{SO}$ de las fórmulas de segundo orden cuyas variables relacionales se encuentran cuantificadas existencialmente, *captura* a la clase **NP**. Stockmeyer [24] generalizó este resultado al definir nuevas clases de complejidad cuya unión estaría caracterizada por toda la lógica de segundo orden y a la cual denominó *Jerarquía Polinomial*. Medina [18] demostró que la subclase de problemas **NP**-completos también se puede caracterizar sintácticamente restringiendo la completitud a reducciones de primer orden proyectivas. Este resultado fue generalizado por Borges y Bonet [5] para cualquier clase de complejidad *buena* [1], entre los que se incluyen los distintos niveles de la Jerarquía Polinomial.

El propósito de este trabajo es realizar un estudio exhaustivo de la clase Σ_2^{p}, el segundo nivel de la Jerarquía Polinomial, explorando desde las distintas propiedades computacionales que la definen, hasta la verificación de varias propiedades sintácticas aplicables en esta clase. La primera parte abarca varios aspectos que se suelen encontrar en la bibliografía, muchas veces a modo de ejercicio. En la parte sintáctica nos concentramos en nuevos métodos para demostrar completitud.

Se ha organizado el trabajo de la siguiente manera:

- En el primer capítulo se muestran los distintos temas que conforman el estudio de la Complejidad Descriptiva. Lo aprovechamos para establecer notaciones y convenciones que se usarán a lo largo del trabajo y para introducir algunos problemas computacionales. Varios conceptos relacionados con computabilidad, como *reducibilidad* y *completitud*, se definen en la última sección, pues es en el ámbito de la Complejidad Descriptiva que nos resultan fundamentales estas ideas.
- En el segundo capítulo se introduce a la Jerarquía Polinomial, donde enfatizamos la subclase Σ_2^p. Los resultados que se muestran se pueden hallar en varios textos (demostrados o propuestos como ejercicios), por lo que este capítulo sirve como un compendio de los aspectos básicos relacionados con esta clase de complejidad. En la última sección, se utiliza la caracterización sintáctica de Σ_2^p para introducir varios problemas importantes.
- En el tercer capítulo se demuestra la completitud vía proyecciones de varios problemas mencionados en el capítulo anterior. El primer teorema es una demostración general de completitud, indispensable para la aplicación de otras propiedades que se mostrarán más adelante. Se complementa el capítulo con una serie de ejemplos.
- En el cuarto capítulo se introduce la teoría relacionada con un nuevo método para demostrar completitud denominado *superfluidad*. Se demuestra que este método se puede aplicar en Σ_2^p, en Π_2^p, junto con varias aplicaciones.
- En el quinto capítulo se hace un recuento de los resultados obtenidos en este trabajo junto a algunas posibles investigaciones que se desprenden o que complementarían esta monografía.

Se anexa además un apéndice listando todos los problemas mencionados a lo largo del trabajo.

Capítulo 1

Preliminares

Repasaremos algunos aspectos básicos de la teoría de la computabilidad y de la lógica matemática. Lo expuesto en este capítulo se encuentra más desarrollado en [2, 13, 20] y [21], pero aprovechamos para destacar los puntos que serán recurrentemente mencionados en los capítulos siguientes. El repaso incluye lógica booleana, lógica de primer y segundo orden, teoría de complejidad computacional y teoría descriptiva de la computabilidad. Algunos de los resultados mostrados son ejercicios propuestos en [13, 21].

1.1. Lógica Booleana

Los problemas relacionados con lógica Booleana han sido reiteradamente utilizados para obtener de un modo natural nuevas propiedades concernientes a la rama de la computación, aplicándose tanto en la producción de prototipos digitales como en la formulación teórica de la computabilidad. En este trabajo nos interesa estudiar la lógica Booleana más como un sistema algebraico que como un sistema formal de razonamiento. Haremos énfasis en la efectividad con que se puede responder algunas preguntas habituales de esta teoría.

Definición 1.1. Una *variable Booleana* es aquella que sólo puede tomar uno de dos valores posibles: **true** o **false**. Entretanto, identificaremos a estas variables por las letras x, y o z y en caso de ser necesario utilizaremos subíndices: $x_1, x_2, x_3, \ldots, x_n$. Las *fórmulas Booleanas* se definen inductivamente:

1. Toda variable Booleana es una fórmula;
2. Si ϕ es una fórmula booleana, entonces $\neg\phi$ es una fórmula Booleana a la cual llamamos *negación* de ϕ;
3. Si ϕ_1 y ϕ_2 son fórmulas Booleanas entonces $\phi_1 \wedge \phi_2$ es una fórmula Booleana a la cual llamamos *conjunción* de ϕ_1 y ϕ_2;
4. Si ϕ_1 y ϕ_2 son fórmulas Booleanas entonces $\phi_1 \vee \phi_2$ es una fórmula Booleana a la cual llamamos *disyunción* de ϕ_1 y ϕ_2.

La notación $\phi(x_1, \ldots, x_n)$ indica que las variables Booleanas $x_1, \ldots, x_n$ ocurren en la fórmula Booleana ϕ. Cuando la cantidad de variables no sea indispensable o se sobreentienda, denotaremos $\phi(\overline{x})$, donde $\overline{x}$ es la tupla de variables Booleanas que aparecen en ϕ. Los siguientes son ejemplos de fórmulas Booleanas:

$$\phi_1(x, y) = (x \vee \neg y)$$
$$\phi_2(x, y, z) = (((x \vee \neg z) \wedge (\neg y \vee x \vee \neg z)) \wedge (\neg x \vee y))$$
$$\phi_3(x, y, z) = ((x \wedge y \wedge z) \vee ((\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge z)))$$
$$\phi_4(x_1, x_2, x_3, x_4) = (((x_1 \wedge x_2) \vee x_3) \wedge (\neg x_4 \vee x_3))$$

A los símbolos $\neg$, $\wedge$ y $\vee$ se les suele llamar *operadores o conectores Booleanos*. Otros operadores Booleanos que no usaremos en esta sección pero que serán recurrentes en todo el trabajo son los llamados *condicionales*. La expresión $(\phi_1 \to \phi_2)$, leída como "ϕ_1 implica ϕ_2", es otra manera de denotar a la fórmula $(\neg\phi_1 \vee \phi_2)$. La expresión $(\phi_1 \leftrightarrow \phi_2)$, leída como "$\phi_1$ si y sólo si ϕ_2", es otra manera de denotar a la fórmula $((\phi_1 \to \phi_2) \wedge (\phi_2 \to \phi_1))$.

Al comienzo de la Definición 1.1 se dijo que una variable Booleana toma uno de dos valores posibles. Si se fija un valor para cada variable que aparece en una fórmula Booleana, esta asignación induce un valor **true** o **false** a la fórmula, de la siguiente manera:

Definición 1.2. Sea $\phi(x_1, \ldots, x_n)$ una fórmula Booleana y $S \in \{\mathbf{true}, \mathbf{false}\}^n$ una asignación de valores de las variables $x_1, \ldots, x_n$. Decimos que S *satisface* ϕ si

1. ϕ es de la forma $\neg\phi_1$ y S no satisface ϕ_1;
2. ϕ es de la forma $(\phi_1 \wedge \phi_2)$ y S satisface ϕ_1 y ϕ_2;
3. ϕ es de la forma $(\phi_1 \vee \phi_2)$ y S satisface ϕ_1 o ϕ_2.

Por ejemplo, la asignación $S = (\mathbf{true}, \mathbf{true}, \mathbf{false})$ satisface la fórmula $\phi_2(x, y, z)$ pero no satisface $\phi_3(x, y, z)$. También diremos que S satisface $\phi_1(x, y)$, aún cuando la variable z no aparezca en esta fórmula. La definición anterior representa el eje central de la lógica Booleana, pues nos permite operar algebraicamente sobre el conjunto de fórmulas Booleanas. La Definición 1.1 sólo nos indica cuándo una secuencia de símbolos representa correctamente una fórmula Booleana, es decir, sólo se refiere a su sintaxis, mientras que la Definición 1.2 proporciona una interpretación a todos los elementos que aparecen en tal secuencia. Bajo

este contexto podemos afirmar que la conjunción $(\phi_1 \wedge \phi_2)$ es "igual" a $(\phi_2 \wedge \phi_1)$, además de muchas otras reglas conocidas.

DEFINICIÓN 1.3. Sean ϕ_1 y ϕ_2 un par de fórmulas Booleanas. Decimos que ϕ_1 y ϕ_2 son *equivalentes*, y lo denotamos por $(\phi_1 \equiv \phi_2)$, si para toda asignación de valores S (común a ambas fórmulas) se cumple que S satisface ϕ_1 si y sólo si S satisface ϕ_2.

PROPOSICIÓN 1.1. Sean ϕ_1, ϕ_2 y ϕ_3 fórmulas Booleanas arbitrarias. Entonces:

1. $\neg\neg\phi_1 \equiv \phi_1$ (doble negación);
2. $(\phi_1 \wedge \phi_1) \equiv \phi_1$ (idempotencia de la conjunción);
3. $(\phi_1 \vee \phi_1) \equiv \phi_1$ (idempotencia de la disyunción);
4. $(\phi_1 \wedge \phi_2) \equiv (\phi_2 \wedge \phi_1)$ (conmutatividad);
5. $(\phi_1 \vee \phi_2) \equiv (\phi_2 \vee \phi_1)$ (conmutatividad);
6. $((\phi_1 \wedge \phi_2) \wedge \phi_3) \equiv (\phi_1 \wedge (\phi_2 \wedge \phi_3))$ (asociatividad);
7. $((\phi_1 \vee \phi_2) \vee \phi_3) \equiv (\phi_1 \vee (\phi_2 \vee \phi_3))$ (asociatividad);
8. $\neg(\phi_1 \wedge \phi_2) \equiv (\neg\phi_1 \vee \neg\phi_2)$ (ley de De Morgan 1);
9. $\neg(\phi_1 \vee \phi_2) \equiv (\neg\phi_1 \wedge \neg\phi_2)$ (ley de De Morgan 2);
10. $((\phi_1 \wedge \phi_2) \vee \phi_3) \equiv ((\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3))$ (distribución);
11. $((\phi_1 \vee \phi_2) \wedge \phi_3) \equiv ((\phi_1 \wedge \phi_3) \vee (\phi_2 \wedge \phi_3))$ (distribución).

De ahora en adelante, cuando presentemos una fórmula Booleana explícitamente usaremos el símbolo de equivalencia ($\equiv$) en vez del símbolo de igualdad. Como consecuencia de las leyes de asociatividad, se omitirán algunos paréntesis internos para mayor comodidad. Además, todos los paréntesis externos que no influyen en la correcta comprensión de las fórmulas planteadas serán también omitidos. Por ejemplo,

$$\phi_1(x, y) \equiv x \vee \neg y$$

$$\phi_2(x, y, z) \equiv (x \vee \neg z) \wedge (\neg y \vee x \vee \neg z) \wedge (\neg x \vee y)$$

$$\phi_3(x, y, z) \equiv (x \wedge y \wedge z) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge z)$$

$$\phi_4(x_1, x_2, x_3, x_4) \equiv ((x_1 \wedge x_2) \vee x_3) \wedge (\neg x_4 \vee x_3)$$

Según la PROPOSICIÓN 1.1, la fórmula ϕ_4 puede ser reescrita como

$$\phi_4(x_1, x_2, x_3, x_4) \equiv (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_4 \vee x_3).$$

Esta nueva forma de $\phi_4(x_1, x_2, x_3, x_4)$ se asemeja mucho a la que posee $\phi_2(x, y, z)$. En adelante, trabajaremos con clases especiales de fórmulas Booleanas.

DEFINICIÓN 1.4. Un *literal positivo* es una variable Booleana y un *literal negativo* es la negación de una variable Booleana. Cuando no sea relevante el signo, diremos simplemente *literal*. Una *cláusula* es una disyunción de literales y un *implicante* es una conjunción de literales. Una fórmula Booleana que sea una conjunción de cláusulas se dice que está en *forma normal conjuntiva*, o FNC para recortar. Una fórmula Booleana que sea una disyunción de implicantes se dice que está en *forma normal disyuntiva*, o FND para recortar.

La fórmula $\phi_2(x, y, z)$ está en FNC y la fórmula $\phi_3(x, y, z)$ está en FND. En el caso de la fórmula $\phi_4(x_1, x_2, x_3, x_4)$, que no estaba originalmente en FNC ni en FND, al aplicarle ciertas operaciones adecuadamente la pudimos transformar a una fórmula equivalente en FNC. De hecho, las propiedades de la PROPOSICIÓN 1.1 permiten demostrar la siguiente:

PROPOSICIÓN 1.2. Toda fórmula Booleana es equivalente a una en FNC y a una en FND.

En general, una tal transformación podría resultar en una fórmula con una cantidad exponencial de símbolos extras en comparación a la fórmula original, lo cual no parece muy conveniente en términos de eficiencia, pero ciertos aspectos de la computabilidad nos permitirán restringirnos particularmente a las clases de fórmulas Booleanas que se encuentran en FNC o en FND.

Ya hemos dado los recursos suficientes para formular problemas de mayor interés. La DEFINICIÓN 1.2 da las condiciones para que una fórmula Booleana se considere con valor **true**, lo cual sucede si existe una asignación de valores que la satisface. En este caso diremos que la fórmula Booleana es *satisfactible*. Se plantea entonces la siguiente pregunta: dada una fórmula Booleana ϕ, ¿es ϕ satisfactible?

Tratemos de responder esta pregunta por partes. Si ϕ es un literal, entonces en efecto es satisfactible. Si ϕ es una cláusula, basta con asignar el valor **true** a alguno de sus literales. Si ϕ es un implicante, a todos sus literales se les debe asignar el valor **true**. En este último caso surge una caracterización interesante: un implicante es satisfactible si y sólo si no contiene un literal y su negación, pues la DEFINICIÓN 1.2 impone que un literal (o una fórmula Booleana, para ser más precisos) y su negación no pueden tener el mismo valor.

Ahora, si la fórmula ϕ está en FNC, ella es satisfactible si y sólo si existe una asignación de valores que proporciona un literal con valor **true** por cada cláusula. Si ϕ está en FND, simplemente debemos garantizar que alguno de sus implicantes sea satisfactible. En el caso de fórmulas en FND la propiedad de satisfabilidad es explícita, es decir, las características que se visualizan en una fórmula en FND son lo suficientemente claras para decidir si la fórmula es satisfactible o no, mientras que para fórmulas en FNC aún no se conoce una caracterización igual de sencilla. En pocas palabras, decidir satisfabilidad para fórmulas en FNC es aparentemente más difícil que en FND. Estas son el tipo de ideas que fomentaron el estudio formal de la computabilidad.

Veamos otro ejemplo. De la DEFINICIÓN 1.3 surge la siguiente pregunta: dadas un par de fórmulas Booleanas ϕ_1 y ϕ_2, ¿son ambas equivalentes? Reescribamos la propiedad de equivalencia. Verificar si $(\phi_1 \equiv \phi_2)$ es lo mismo que verificar si la fórmula Booleana

$$\phi \equiv (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)$$

es siempre satisfactible, es decir, que toda asignación de valores apropiada a la fórmula ϕ sea un certificado. Una fórmula Booleana que cumpla tal propiedad se dice que es una *tautología*. Sin muchos detalles, que ϕ sea tautología quiere decir que ϕ_1 y ϕ_2 son ambas **true** o son ambas **false** para toda asignación de valores apropiada a ϕ, y esta es justamente la DEFINICIÓN 1.3. Hemos reducido el planteamiento anterior al siguiente: dada una fórmula Booleana ϕ, ¿es ϕ una tautología? Una vez más, nos valemos de la PROPOSICIÓN 1.2 para dar una respuesta a esta pregunta por partes.

Una cláusula ϕ será tautología siempre que contenga un literal y su negación. Si ϕ es un implicante no hay nada interesante que decir. Si ϕ está en FNC entonces basta con verificar si alguna de sus cláusulas es tautología. Si ϕ está en FND se debe verificar que toda asignación apropiada a ϕ certifica al menos uno de sus implicantes. Se generó una situación similar a la del problema de satisfabilidad: tautología para fórmulas en FNC es explícito, mientras que para fórmulas en FND se requieren mayores esfuerzos.

Otra modo de obtener el problema de tautología es mediante el problema de satisfabilidad junto con propiedades algebraicas. Por la DEFINICIÓN 1.2 tenemos que una asignación de valores que satisface una fórmula Booleana ϕ no puede satisfacer a $\neg\phi$ y de aquí es directo que si ϕ no es satisfactible, entonces $\neg\phi$ debe ser una tautología. Por las leyes de De Morgan,

si una fórmula Booleana ϕ está en FNC (resp. en FND) entonces $\neg\phi$ está en FND (resp. en FNC). El problema de satisfabilidad induce naturalmente la pregunta del problema de *insatisfabilidad* (no satisfabilidad), el cual es análogo al problema de tautología.

Resumiendo las ideas previas, insatisfabilidad para fórmulas en FNC es tan difícil de decidir como tautología para fórmulas en FND, mientras que insatisfabilidad para fórmulas en FND es tan claro como tautología para fórmulas en FNC.

Hasta este momento sólo hemos introducido variantes de problemas computacionales y clasificándolos como complicados o evidentes según las caracterizaciones que se puedan deducir, pero estamos especialmente interesados en estudiar estas herramientas de clasificación formalmente, que es justamente en lo que se basa la Teoría de Complejidad Computacional. Antes de hablar en estos términos, es necesario introducir un tipo de lenguaje apropiado.

1.2. Lógica de Primer Orden

Mientras que la lógica Booleana sólo la estamos usando para motivar las ideas básicas de la computabilidad, la lógica de primer orden es el sistema formal de lenguaje que usamos para describir gran parte de los resultados. En cierto sentido, la lógica de primer orden (también llamada lógica de predicados) extiende a la lógica Booleana al agregarle a esta última un mayor poder expresivo [**20**]. Todo lo relacionado con la estructuración algebraica de las expresiones que manejaremos a continuación seguirán satisfaciendo las propiedades mencionadas en las Proposiciones 1.1 y 1.2, las cuales en algunos casos usaremos a nuestra conveniencia. Otros aspectos son nuevos en comparación con la lógica anterior, muy particularmente la herramienta de cuantificación.

Definición 1.5. Un *vocabulario relacional* es una tupla de símbolos $\langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$, donde cada a_i es un número entero positivo fijo, cada R_i se dice que es un *símbolo de relación de aridad* a_i y cada c_j se dice que es un *símbolo de constante*. Sea $\mathrm{Var} = \{x_1, x_2, x_3, \ldots\}$ un conjunto numerable de variables, sobre un dominio por definir (estas variables no son variables Booleanas). Un *término* es o un elemento de Var o un símbolo de constante.

En general, un vocabulario también se puede componer de símbolos de funciones, pero éstos no se suelen utilizar en teoría de la computabilidad. No existe mayor riesgo al omitirlos, pues toda función se puede ver como una relación cuando la asociamos a su imagen.

Los símbolos de funciones también inducen *términos* de manera recursiva, por lo que esta terminología también difiere de la usual por no considerarse símbolos de funciones.

A los símbolos de relación también se les suele llamar *predicados* y aquí también adoptamos esa terminología. Cada vocabulario induce un conjunto de fórmulas que, según el modelo matemático con que trabajemos, son capaces de expresar un sinfín de propiedades.

DEFINICIÓN 1.6. Sea σ un vocabulario y R un predicado de σ de aridad a. Una expresión de la forma $R(t_1, \ldots, t_a)$, donde $t_1, \ldots, t_a$ son términos, se denomina *fórmula atómica*. Las *fórmulas de primer orden sobre el vocabulario* σ se definen inductivamente:

1. Toda fórmula atómica de σ es una fórmula de primer orden;
2. Si φ es una fórmula de primer orden, entonces $\neg\varphi$ es una fórmula de primer orden;
3. Si φ_1 y φ_2 son fórmulas de primer orden, entonces $\varphi_1 \wedge \varphi_2$ y $\varphi_1 \vee \varphi_2$ son fórmulas de primer orden;
4. Si φ es una fórmula de primer orden y x es una variable, entonces $\exists x\varphi$ y $\forall x\varphi$ son fórmulas de primer orden.

A una fórmula atómica o a la negación de una fórmula atómica también se le llama *literal*, y se dice que es positivo o negativo según el signo. El *lenguaje de primer orden sobre* σ es el conjunto $\mathbf{FO}[\sigma]$ de todas las fórmulas de primer orden sobre σ. La lógica de primer orden $\mathbf{FO}$ se define como la unión de todos los lenguajes $\mathbf{FO}[\sigma]$.

Los conectores Booleanos nos permitirán utilizar los símbolos $\rightarrow$ y $\leftrightarrow$ como se indicó en la sección anterior. Al símbolo $\exists$ lo llamamos *cuantificador existencial* y a $\forall$ *cuantificador universal*. Por comodidad simplificamos una posible secuencia $\exists x \exists y \varphi$ por $\exists xy\varphi$.

Hasta ahora sólo hemos hecho referencia a las reglas sintácticas que rigen a las fórmulas de primer orden, pero así como la asignación de valores otorga a las fórmulas Booleanas y a sus símbolos una interpretación, también podemos dotar de una interpretación adecuada a las fórmulas de primer orden. Teniendo en cuenta las similitudes que tienen ambas lógicas, necesitamos un modelo que asigne un valor **true** o **false** a cada fórmula atómica y que estos valores induzcan un valor de verdad a las fórmulas de primer orden en general.

DEFINICIÓN 1.7. Sea el vocabulario $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. Una σ-*estructura* es una tupla $\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$, donde:

1. $|\mathcal{A}|$ es un conjunto no vacío denominado *dominio* de $\mathcal{A}$;

2. $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$, para todo $i = 1, \ldots, r$;
3. $c_j^{\mathcal{A}} \in |\mathcal{A}|$, para todo $j = 1, \ldots, s$.

Cuando el vocabulario no sea relevante diremos simplemente que $\mathcal{A}$ es una *estructura*.

Todos las estructuras que se usan en este trabajo son con dominio finito, por lo que no hará falta hacer la distinción en ningún momento. La cardinalidad de $|\mathcal{A}|$ es el *tamaño* de la estructura $\mathcal{A}$. Denotamos por STRUC[σ] al conjunto de todas las σ-estructuras. Se considerarán dominios ordenados y asumimos de ahora en adelante que todos los dominios son de la forma $|\mathcal{A}| = [n] = \{0, 1, \ldots, n-1\}$ con el orden usual de los enteros. En caso de no tener explícitamente esa forma se especificará cuál es el orden. La relación de orden no es la única que se considerará subyacente a todas las estructuras. Enlistamos las relaciones y constantes que forman parte de todas las estructuras que consideraremos y que por tanto no será necesario incluirlas como símbolos de vocabularios:

1. La relación binaria de igualdad, denotada por '=' como es usual.
2. La relación binaria de orden total, denotada por '$\leq$' como es usual.
3. La relación binaria suc, que para toda estructura $\mathcal{A}$ el conjunto $\text{suc}^{\mathcal{A}}$ posee el par (i, j) siempre que j sea el sucesor de i bajo el orden total.
4. La relación binaria bit, que para toda estructura $\mathcal{A}$ con dominio $[n]$ el conjunto $\text{bit}^{\mathcal{A}}$ posee el par (i, j) siempre que el j-ésimo bit de la representación binaria de i sea 1.
5. La constante 0, que representa el mínimo elemento del dominio bajo el orden total.
6. La constante máx, que representa el máximo elemento del dominio bajo el orden total.

Note que todas estas relaciones y constantes no dependen de cómo sea $\mathcal{A}$ estructuralmente, sino del conjunto $[n]$ en sí y es por ello que reciben el nombre de *relaciones y constantes numéricas*. Por ejemplo, para todas las estructuras $\mathcal{A}$ con dominio $[n]$ siempre será cierto que los pares de la forma $(i, i+1)$ serán elementos de $\text{suc}^{\mathcal{A}}$, para todo $i = 0, \ldots, n-2$. La relación bit funciona como una base de datos que registra las codificaciones binarias de los números enteros. Otro aspecto que asumimos como cierto es que 0 y máx nunca representan un mismo elemento, por lo que siempre trabajaremos con estructuras de tamaño 2 como mínimo. Dado un vocabulario σ, denotamos por $\mathcal{L}[\sigma]$ al lenguaje de fórmulas de primer orden sobre σ junto con las relaciones y constantes numéricas. Mostramos ahora qué significa que una fórmula φ de $\mathcal{L}[\sigma]$ sea cierta, dada una σ-estructura $\mathcal{A}$.

DEFINICIÓN 1.8. Sea el vocabulario $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$, $\mathcal{A}$ una σ-estructura y φ una fórmula de $\mathcal{L}[\sigma]$. Consideremos una función $\iota : \text{Var} \to |\mathcal{A}|$. Por conveniencia, digamos que $\iota(c_j) = c_j^{\mathcal{A}}$ para todo $j = 1, \ldots, s$. A ι la denominamos *interpretación*. Describiremos qué significa que $\mathcal{A}$, junto con la interpretación ι, *satisfaga* la fórmula φ, denotándolo por $(\mathcal{A}, \iota) \models \varphi$.

1. Si φ es la fórmula atómica $R_i(t_1, \ldots, t_{a_i})$, con $i = 1, \ldots, r$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si $(\iota(t_1), \ldots, \iota(t_{a_i})) \in R_i^{\mathcal{A}}$.
2. Si φ es de la forma $\neg\varphi_1$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si no es el caso que $(\mathcal{A}, \iota) \models \varphi_1$ (esto último se denota por $(\mathcal{A}, \iota) \not\models \varphi_1$).
3. Si φ es de la forma $(\varphi_1 \wedge \varphi_2)$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si $(\mathcal{A}, \iota) \models \varphi_1$ y $(\mathcal{A}, \iota) \models \varphi_2$.
4. Si φ es de la forma $(\varphi_1 \vee \varphi_2)$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si $(\mathcal{A}, \iota) \models \varphi_1$ ó $(\mathcal{A}, \iota) \models \varphi_2$.
5. Si φ es de la forma $\exists x \varphi_1$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si $(\mathcal{A}, \iota_{x=a}) \models \varphi_1$ para algún $a \in |\mathcal{A}|$. La interpretación $\iota_{x=a}$ es la misma ι salvo que la variable x se reinterpreta con el valor a.
6. Si φ es de la forma $\forall x \varphi_1$, entonces $(\mathcal{A}, \iota) \models \varphi$ si y sólo si $(\mathcal{A}, \iota_{x=a}) \models \varphi_1$ para todo $a \in |\mathcal{A}|$.

Estas últimas propiedades relacionadas con los cuantificadores implican que si $\varphi \in \mathcal{L}[\sigma]$ es una sentencia y $\mathcal{A}$ una σ-estructura entonces que $\mathcal{A}$ satisfaga φ o no es independiente de la interpretación que se escoja, puesto que al fin y al cabo todas las variables involucradas serán reinterpretadas. Es por ello que en el caso de sentencias omitimos el uso de interpretaciones y replanteamos la notación. Si φ es una sentencia, denotamos $\mathcal{A} \models \varphi$ para indicar que $\mathcal{A}$ satisface la fórmula φ, y en caso de no satisfacerla, denotamos $\mathcal{A} \not\models \varphi$.

La DEFINICIÓN 1.8 es un concepto preciso de la idea de asignación de valor en este lenguaje formal. En este sentido podemos hablar también de equivalencia de fórmulas de primer orden, como en la DEFINICIÓN 1.3.

DEFINICIÓN 1.9. Sea σ un vocabulario y $\varphi_1, \varphi_2 \in \mathcal{L}[\sigma]$ un par de sentencias. Decimos que φ_1 y φ_2 son *equivalentes*, denotado por $\varphi_1 \equiv \varphi_2$, si para toda σ-estructura $\mathcal{A}$ se cumple que $\mathcal{A} \models \varphi_1$ si y sólo si $\mathcal{A} \models \varphi_2$.

Por ejemplo, $\forall x \varphi$ es una expresión equivalente a la fórmula $\neg\exists x \neg\varphi$. A $\forall$ se le denomina *cuantificador universal*. Todas las variables que están al alcance de un cuantificador en una

fórmula de primer orden se dice que son *variables cuantificadas*, de lo contrario se dice que son *variables libres*. La notación $\varphi(x_1, \ldots, x_m)$ indica que las variables libres de φ son elementos de $\{x_1, \ldots, x_m\}$. Se dice que $\varphi(x_1, \ldots, x_m)$ es una *fórmula sin variables libres*, o una *sentencia*, si todas las variables $x_1, \ldots, x_m$ se encuentran cuantificadas. En adelante, cuando presentemos una fórmula de primer orden explícitamente usaremos el símbolo de equivalencia ($\equiv$).

Ejemplo 1.1. A una fórmula que sólo se expresa con relaciones numéricas la llamamos *fórmula numérica*, y la verdad o falsedad de tales fórmulas sólo dependerá del conjunto $[n]$. Deduzcamos algunas propiedades en función de las relaciones numéricas que ya conocemos.

La relación de orden total $\leq$, implícita sobre cualquier estructura $\mathcal{A}$, permite definir o representar otras relaciones de interés. Supongamos que $\mathcal{A}$ tiene dominio $[n]$, entonces la relación binaria $<$, llamada *orden estricto*, se define como

$$(x < y) \equiv (x \leq y) \wedge (x \neq y), \tag{1.1}$$

para todo par de elementos $x, y \in [n]$. La relación sucesor puede reescribirse como

$$\mathrm{suc}(x, y) \equiv (x < y) \wedge \neg\exists z\big((x < z) \wedge (z < y)\big), \tag{1.2}$$

para todo par de elementos $x, y \in [n]$. También podemos extender las relaciones numéricas a otros dominios. Sea $k \geq 2$ un número natural y consideremos el conjunto de k-tuplas, $[n]^k$. En $[n]^k$ se define inductivamente la relación $\leq_k$ como sigue:

$$(x_1, \ldots, x_k) \leq_k (y_1, \ldots, y_k) \equiv x_1 < y_1 \vee \big(x_1 = y_1 \wedge (x_2, \ldots, x_k) \leq_{k-1} (y_2, \ldots, y_k)\big),$$

donde $\leq_1$ es el orden de $[n]$. A $\leq_k$ lo llamamos *orden lexicográfico* u *orden del diccionario* y es en efecto un orden total sobre el conjunto $[n]^k$. $[n]^k$ con este orden es *isomorfo* a $[n^k]$ con el orden usual. Es decir, existe una función biyectiva $f : [n]^k \to [n^k]$ tal que para todos $\overline{x}, \overline{y} \in [n]^k$ se cumple que

$$\overline{x} \leq_k \overline{y} \qquad \Rightarrow \qquad f(\overline{x}) \leq f(\overline{y}).$$

Explícitamente, este función es

$$f(x_1, x_2, \ldots, x_k) = x_1 n^{k-1} + x_2 n^{k-2} + \cdots + x_k.$$

En general, se dice que dos estructuras son isomorfas si existe una biyección entre ellas que preserva todas las relaciones subyacentes.

En $[n]^k$ podemos considerar también el orden estricto $<_k$ y la relación sucesor suc_k, definiéndolas con unas fórmulas análogas a (1.1) y (1.2).

También podemos determinar en primer orden si un número x en $[n]$ es par o impar. Una u otra pregunta se pueden responder con las siguientes fórmulas:

$$\mathrm{par}(x) \equiv \neg\,\mathrm{bit}(x, 0) \quad \text{y} \quad \mathrm{impar}(x) \equiv \neg\mathrm{par}(x).$$

Otra relación que nos interesa es $\log(x, y)$ que determina si y es el menor entero por encima del logaritmo de x, es decir, si es cierto que $y = \lceil \log_2(x) \rceil$. Esto se puede responder con la siguiente fórmula numérica:

$$\log(x, y) \equiv \mathrm{bit}(x, y) \wedge \forall z\big(y < z \to \neg\,\mathrm{bit}(x, z)\big). \tag{1.3}$$

En efecto, el valor de y coincide con el menor bit de x cuyo valor es 1. Si $x = 0$ simplemente imponemos $y = 0$. Cada vez que se defina un dominio asumiremos que la relación bit ya está a disposición. No se debe confundir esta relación con la capacidad de construir arbitrariamente una secuencia binaria. Esto último lo estudiaremos en un sentido estructural. □

En los ejemplos que siguen se fijarán varias características que consideramos pertinentes mantener a lo largo del trabajo.

Ejemplo 1.2. Un *código binario* es una secuencia finita de ceros y unos. Tales secuencias las vamos a considerar en un contexto estructural.

Con un dominio de tamaño n podemos codificar hasta 2^n estructuras distintas, y cada una de estas estructuras está asociada unívocamente a los números enteros entre 0 y 2^n-1, pues se trata simplemente de la codificación binaria de cada uno de ellos en n bits, donde un *bit* es un elemento del dominio $[n]$. El orden natural de $[n]$ es inverso al orden de los bits de un código binario, tal como sucede cuando escribimos un número en base 10, base 2 o cualquier otra. Desde un punto de vista estructural, un código binario es simplemente el reconocimiento de aquellos bits donde hay un 1. Consideremos el vocabulario $\sigma_s = \langle S^1 \rangle$. Asociamos la secuencia binaria 10001101 con la σ_s-estructura $\mathcal{A} = \langle [8], S^{\mathcal{A}} \rangle$, donde $S^{\mathcal{A}} = \{0, 2, 3, 7\}$. Cada subconjunto de $[n]$ define una σ_s-estructura distinta, quedando justificada así la afirmación inicial, pues hay 2^n subconjuntos distintos de $[n]$.

Más adelante veremos que cualquier σ-estructura la podemos interpretar eficientemente como un código binario. □

EJEMPLO 1.3. Un *grafo* es un par $\mathcal{G} = \langle V, A\rangle$, donde V es un conjunto no vacío, cuyos elementos se llaman *vértices* y A es un subconjunto de V^2, cuyos elementos se llaman *aristas*. En el contexto de la teoría de modelos, un grafo no es más que una estructura sobre el vocabulario $\sigma_g = \langle A^2\rangle$. Muchas de las propiedades usuales de grafos se pueden describir mediante fórmulas en $\mathcal{L}[\sigma_g]$. Por ejemplo, $\forall x\neg A(x,x)$ describe la propiedad de *grafo sin lazos*, es decir, sin aristas cuyo vértice de salida sea igual al de llegada. En adelante, asumimos que todos los grafos satisfacen esta propiedad.

Un *grafo no dirigido* es aquel cuyas aristas se definen como subconjuntos de V de tamaño 2 (o menor que 2 si aceptáramos lazos) pero esta propiedad la podemos describir con la siguiente fórmula:

$$\forall xy\big(A(x,y) \to A(y,x)\big),$$

es decir, un grafo no dirigido se puede interpretar como una σ_g-estructura $\mathcal{G}$ tal que $A^{\mathcal{G}}$ satisface la propiedad de simetría. Asumimos también que todas las σ_g-estructuras satisfacen implícitamente esta propiedad. En términos de eficiencia, esto no representa ningún problema. Por ejemplo, si se da el caso de que un grafo no dirigido $\mathcal{G}$ se quiere redefinir sin una arista $\{x,y\}$, basta con que a $A^{\mathcal{G}}$ le quitemos los pares (x,y) y (y,x). Y cuando digamos que un par (x,y) está en $A^{\mathcal{G}}$ asumimos directamente que (y,x) también lo está.

Un grafo es *completo* si todo par de vértices define una arista del grafo. En lenguaje de primer orden, un grafo es completo si satisface

$$\forall xy\big(x \neq y \to A(x,y)\big).$$

Note que todas las fórmulas planteadas son de hecho sentencias. En general, este es el tipo de fórmulas con las que trabajaremos. □

EJEMPLO 1.4. No siempre una estructura se define a través de vocabularios, recuérdense las fórmulas Booleanas, que presentamos inicialmente como simples secuencias de símbolos. El lenguaje de predicados es lo suficientemente bueno para describir palabras o secuencias finitas, siempre que éstas se rijan por una cantidad finita de condiciones estructurales, como sucede con las reglas sintácticas que actúan sobre la correcta estructuración de una fórmula Booleana.

Trabajemos primero con la clase de fórmulas Booleanas en FNC y consideremos el vocabulario $\sigma_{\text{fnc}} = \langle P^2, N^2\rangle$. A cada fórmula Booleana ϕ en FNC le asociamos una σ_{fnc}-estructura

$\mathcal{A}_\phi$. El propósito de los símbolos P y N es bastante claro: P permite identificar los literales que aparecen positivos en las cláusulas de ϕ y N hace lo propio con los literales negativos. El dominio requiere un poco más de rigor. Podemos definir $|\mathcal{A}_\phi| = [n]$, donde n es el máximo entre variables Booleanas y número de cláusulas de ϕ. Por conveniencia, imponemos que el número n siempre coincida con la segunda condición.

Consideremos la fórmula $\phi_2(x, y, z)$ de la sección anterior. La estructura que le corresponde es $\mathcal{A}_{\phi_2} = \langle [3], P^{\mathcal{A}_{\phi_2}}, N^{\mathcal{A}_{\phi_2}} \rangle$, donde

$$P^{\mathcal{A}_{\phi_2}} = \{(0,0), (1,0), (2,1)\} \quad \text{y} \quad N^{\mathcal{A}_{\phi_2}} = \{(0,2), (1,1), (1,2), (2,0)\}.$$

Los elementos 0, 1 y 2 se refieren a las variables x, y y z respectivamente, y a la vez se refieren a las 3 cláusulas de $\phi_2(x, y, z)$ en el orden en que aparecen. Un par (i, j) toma como primera coordenada la cláusula a considerar y segunda coordenada la variable Booleana. Responder si (i, j) está en P (o en N) es verificar si la variable j-ésima es positiva (o negativa) en la i-ésima cláusula. Consideremos ahora a $\phi_4(x_1, x_2, x_3, x_4)$ con su representación en FNC obtenida en la sección anterior. Note que el dominio de $\mathcal{A}_{\phi_4}$ debe ser $[4]$, pero ϕ_4 en la representación actual no tiene 4 cláusulas. Por la Proposición 1.1:

$$\phi_4(x_1, x_2, x_3, x_4) \equiv \phi_4(x_1, x_2, x_3, x_4) \wedge (x_4 \vee \neg x_4).$$

Cuando el tamaño del dominio supera a la cantidad de cláusulas agregamos tantas tautologías $(x \vee \neg x)$ como sea necesario. De esta manera, $\mathcal{A}_{\phi_4} = \langle [4], P^{\mathcal{A}_{\phi_4}}, N^{\mathcal{A}_{\phi_4}} \rangle$, donde

$$P^{\mathcal{A}_{\phi_4}} = \{(0,0), (0,2), (1,1), (1,2), (2,2), (3,3)\} \quad \text{y} \quad N^{\mathcal{A}_{\phi_4}} = \{(2,3), (3,3)\}.$$

El mismo vocabulario σ_{fnc} se podría utilizar para dar una representación estructural de fórmulas Booleanas en FND, pero para evitar confusiones nos valemos de otros símbolos relacionales. Usamos el vocabulario $\sigma_{\text{fnd}} = \langle Q^2, M^2 \rangle$, donde Q y M tienen una función similar a las de P y N respectivamente, pero con estos nuevos símbolos se reconocen implicantes en vez de cláusulas. De esta manera, la fórmula $\phi_3(x, y, z)$ de la sección anterior se asocia con la estructura $\mathcal{A}_{\phi_3} = \langle [3], Q^{\mathcal{A}_{\phi_3}}, M^{\mathcal{A}_{\phi_3}} \rangle$, donde

$$Q^{\mathcal{A}_{\phi_3}} = \{(0,0), (0,1), (0,2), (1,1), (1,2), (2,2)\} \quad \text{y} \quad M^{\mathcal{A}_{\phi_3}} = \{(1,0), (2,0), (2,1)\}.$$

De ahora en adelante, cuando nos refiramos a una fórmula Booleana puede ser tanto en el contexto de la Definición 1.1 como de una estructura sobre σ_{fnc} o σ_{fnd}.

Recordemos ahora algunas propiedades que surgieron al final de la sección anterior: satisfabilidad e insatisfabilidad. Cada uno de estos problemas se clasificaron como sencillos o complicados según el tipo de fórmulas Booleanas a utilizar. La primera propiedad resultaba particularmente fácil si nos limitábamos a fórmulas en FND, pues bastaba con reconocer aspectos estructurales de la fórmula propuesta. Sea ϕ una fórmula Booleana en FND y $\mathcal{A}_\phi$ la σ_{fnd}-estructura que la representa. ϕ es satisfactible si y sólo si

$$\mathcal{A}_\phi \models \exists x \forall y \big(Q(x,y) \to \neg M(x,y)\big).$$

La sentencia planteada verifica que en algún implicante de ϕ no haya un literal y su negación. Insatisfabilidad en FND se caracteriza por la negación de la sentencia anterior: ϕ es insatisfactible si y sólo si

$$\mathcal{A}_\phi \models \forall x \exists y \big(Q(x,y) \wedge M(x,y)\big).$$

Para fórmulas en FNC la lógica de primer orden no resulta lo suficientemente fuerte para caracterizar una propiedad que depende de otros agentes además de los estructurales. En [21] se muestran las limitaciones expresivas de la lógica de primer orden como consecuencia de varios teoremas tradicionales de la lógica matemática. En [3] se muestra lo mismo aplicando una técnica elegante conocida como *Juegos de Ehrenfeucht-Fraissé*. No haremos un recuento de esos resultados ya que se alejan del objetivo principal de este trabajo pero sí destacamos, a modo introductorio, la necesidad de un lenguaje con mayor poder expresivo para caracterizar problemas de mayor dificultad. □

1.3. Lógica de Segundo Orden

Para obtener fórmulas de segundo orden extendemos el conjunto Var de variables de primer orden, al conjunto Var_2 de *variables de segundo orden o relacionales* $X_1, X_2, X_3, \ldots$, las cuales simbolizan relaciones de cualquier aridad.

Definición 1.10. Sea σ un vocabulario. Las fórmulas atómicas sobre σ se extienden a aquellas de la forma $X(t_1, \ldots, t_a)$, donde X es una variable relacional de aridad a y $t_1, \ldots, t_a$ son términos, es decir, elementos de Var o constantes de σ. Las *fórmulas de segundo orden sobre el vocabulario* σ se definen inductivamente

1. Toda fórmula atómica de σ es una fórmula de segundo orden;

2. Toda fórmula de primer orden definida a partir de fórmulas atómicas sobre σ es una fórmula de segundo orden;
3. Si Φ es una fórmula de segundo orden y X es una variable relacional de aridad a entonces $\exists X^a \Phi$ y $\forall X^a \Phi$ son fórmulas de segundo orden;
4. Todas aquellas fórmulas que podamos formar con los operadores Booleanos.

A una fórmula atómica o a la negación de una fórmula atómica también se le llama *literal*. El *lenguaje de segundo orden sobre* σ es el conjunto $\mathbf{SO}[\sigma]$ compuesto por todas las fórmulas de segundo orden sobre σ. La lógica de segundo orden $\mathbf{SO}$ se define como la unión de todos los lenguajes $\mathbf{SO}[\sigma]$.

Simplificamos una posible secuencia $\exists X^a \exists Y^b \varphi$ por $\exists X^a Y^b \varphi$. La relación entre los cuantificadores $\exists$ y $\forall$ es análoga a la de primer orden. Una variable relacional que no se encuentre al alcance de un cuantificador se dice que es una variable libre. A una fórmula de segundo orden la solemos denotar con letras griegas mayúsculas: Φ, Θ, Ψ, etc. La notación $\Phi(x_1, \ldots, x_m, X_1, \ldots, X_{m'})$ indica que las variables libres de Φ son elementos de $\{x_1, \ldots, x_m, X_1, \ldots, X_{m'}\}$. Se dice que Φ es una fórmula *sin variables libres* o *sentencia* si todas las variables, de primer y segundo orden, se encuentran cuantificadas en Φ.

Según la Definición 1.10, $\mathbf{FO}[\sigma] \subseteq \mathbf{SO}[\sigma]$ para todo vocabulario σ, y por tanto, $\mathbf{FO} \subseteq \mathbf{SO}$. Esta contención es estricta [**3, 21**] y es la extensión que usaremos para caracterizar problemas más complicados. La verdad o falsedad de una fórmula de segundo orden es a su vez una extensión de la Definición 1.8.

Definición 1.11. Sea σ un vocabulario, $\mathcal{A}$ una σ-estructura y Φ una fórmula de $\mathbf{SO}[\sigma]$. Sea ι una interpretación sobre $\mathcal{A}$ y consideremos una extensión de ι, denotada por $\jmath$, que a cada variable relacional X de aridad a le asigna un subconjunto $\jmath(X) \subseteq |\mathcal{A}|^a$. Describiremos qué significa que $\mathcal{A}$, junto con la interpretación $\jmath$, *satisfaga* la fórmula Φ, denotándolo por $(\mathcal{A}, \jmath) \models \Phi$:

1. Como $\jmath(t) = \iota(t)$, para todo término t, las propiedades de la Definición 1.8 son todas válidas.
2. Si X es una variable relacional y Φ es la fórmula atómica $X(t_1, \ldots, t_a)$, entonces $(\mathcal{A}, \jmath) \models \Phi$ si y sólo si $(\iota(t_1), \ldots, \iota(t_a)) \in \jmath(X)$;

3. Si Φ es de la forma $\exists X^a \varphi$, entonces $(\mathcal{A}, \jmath) \models \Phi$ si y sólo si existe $T \subseteq |\mathcal{A}|^a$ tal que $(\mathcal{A}, \jmath_{X=T}) \models \varphi$. La interpretación $\jmath_{X=T}$ es la misma $\jmath$ salvo que la variable relacional X se reinterpreta como la relación T.

Para el cuantificador universal se tiene que $(\mathcal{A}, \jmath) \models \forall X^a \varphi$ si y sólo si $(\mathcal{A}, \jmath_{X=T}) \models \varphi$ para todo $T \subseteq |\mathcal{A}|^a$. Si Φ es una sentencia, la interpretación $\jmath$ no es indispensable. Si la sentencia Φ es cierta para una estructura $\mathcal{A}$, denotamos $\mathcal{A} \models \Phi$, y en caso de ser falsa, $\mathcal{A} \not\models \Phi$.

Definición 1.12. Sea σ un vocabulario y $\Phi_1, \Phi_2 \in \mathbf{SO}[\sigma]$. Decimos que Φ_1 y Φ_2 son *equivalentes*, denotado por $\Phi_1 \equiv \Phi_2$, si para toda σ-estructura $\mathcal{A}$ se cumple que $\mathcal{A} \models \Phi_1$ si y sólo si $\mathcal{A} \models \Phi_2$.

Como es de esperar, cuando presentemos una fórmula de segundo orden explícitamente usaremos el símbolo de equivalencia ($\equiv$).

Ejemplo 1.5. El problema de satisfabilidad para fórmulas en FNC es expresable en segundo orden. Sea el vocabulario σ_{fnc} y la fórmula en $\mathbf{SO}[\sigma_{\text{fnc}}]$,

$$\Phi_{\text{sat}} \equiv \exists S^1 \forall x \exists y \big((P(x, y) \wedge S(y)) \vee (N(x, y) \wedge \neg S(y)) \big).$$

Considerando las fórmulas Booleanas en FNC como σ_{fnc}-estructuras, la fórmula de segundo orden Φ_{sat} se interpreta de la siguiente manera: existe un conjunto S de variables Booleanas con valor asignado **true**, el cual certifica a la fórmula Booleana en cuestión. Note que esta sentencia no especifica de qué modo podemos hallar o construir el certificado, simplemente introduce una variable relacional S que nos permite caracterizar sintácticamente la propiedad de satisfabilidad en FNC. En el caso de insatisfabilidad, pensamos en la negación de la fórmula anterior:

$$\Phi_{\text{unsat}} \equiv \neg \Phi_{\text{sat}} \equiv \forall S^1 \exists x \forall y \big((P(x, y) \to \neg S(y)) \wedge (N(x, y) \to S(y)) \big).$$

Φ_{sat} pertenece a una clase especial de fórmulas conocido como *fragmento existencial de fórmulas de segundo orden*, denotado por $\exists\mathbf{SO}$. En este conjunto se encuentran todas las sentencias cuyas variables relacionales están cuantificadas existencialmente. Análogamente, podemos definir el *fragmento universal de fórmulas de segundo orden*, denotado por $\forall\mathbf{SO}$, a la cual pertenece Φ_{unsat}. Otra fórmula de interés y perteneciente al fragmento universal es

la relacionada con tautología para fórmulas en FND:

$$\Phi_{\text{tauto}} \equiv \forall T^1 \exists x \forall y \big((Q(x,y) \to T(y)) \wedge (M(x,y) \to \neg T(y)) \big).$$

La sentencia anterior se basa en la reducción natural de fórmulas Booleanas en FNC insatisfactibles a fórmulas Booleanas en FND tautológicas, vista en la primera sección de este capítulo. Los fragmentos $\exists$**SO** y $\forall$**SO** fueron los primeros en ser asociados a procesos computacionales. □

1.4. Complejidad Computacional

En teoría de computabilidad, un *problema de decisión* es una pregunta planteada dentro de los lineamientos de algún sistema formal (en particular, cualquier tipo de teoría matemática) que requiere un " sí " o un " no " como respuesta final. El formato que adoptamos en este trabajo al presentar un problema de decisión es el de **Instancia-Pregunta**, impuesto en [**10**] y usado en otras guías como [**6, 16, 23**]. Por ejemplo:

- **Instancia:** $\mathcal{G} = \langle V, A \rangle$ grafo no dirigido.
  Pregunta: ¿Es $\mathcal{G}$ 2-colorable o bipartito?[1]
- **Instancia:** $\mathcal{G} = \langle V, A \rangle$ grafo no dirigido.
  Pregunta: ¿Es $\mathcal{G}$ un grafo hamiltoniano?[2]
- **Instancia:** $\phi(x_1, \ldots, x_n)$ fórmula booleana.
  Pregunta: ¿Existe $(a_1, \ldots, a_n) \in \{\textbf{true}, \textbf{false}\}^n$ tal que $(a_1, \ldots, a_n) \models \phi$?

Para cada uno de estos problemas existe un algoritmo que determina una respuesta afirmativa o negativa según la instancia que se suministre. Un problema de decisión que tiene una solución algorítmica se dice que es *recursivo*. Todos los problemas mencionados en este trabajo lo son. Los algoritmos permiten clasificar un problema según la cantidad de recursos utilizados, lo cual se puede hacer de un modo bastante preciso una vez establecido un modelo computacional formal que describa la efectividad del algoritmo. Durante la década de 1930 se plantearon varios de estos modelos: λ-cálculo, funciones recursivas de Gödel, algoritmos de Markov, máquinas de Turing, etc. Todos estos modelos se demostraron equivalentes entre sí, lo que supuso la general aceptación de la *tesis de Church*: "Todo concepto intuitivo

[1] Que el conjunto de vértices se particiona en dos bloques y en cada bloque no hay vértices adyacentes.

[2] Que posee un camino que pasa por cada vértice del grafo exactamente una vez.

de algoritmo es capturado apropiadamente por la máquina de Turing". Es por ello que la Máquina de Turing representa el concepto base que rige la idea formal de algoritmo. Intuitivamente, una Máquina de Turing suele visualizarse como se muestra en la FIGURA 1.1. En ella se aprecian los instrumentos que conforman una Maquina de Turing, de los que se dispone como describimos a continuación:

- Una cinta infinita de *celdas*. A partir de una celda inicial, un usuario suministra una *palabra*, o secuencia finita de símbolos, donde cada símbolo ocupa una celda.
- Un *apuntador* que se encarga de leer, borrar o escribir símbolos en las celdas, según las instrucciones que se le dan. El apuntador sólo puede desplazarse una celda a la vez y no puede dar un paso a la izquierda del *símbolo inicial* $\triangleright$. La posición de origen del apuntador es por debajo del símbolo inicial y éste se mueve una vez se haya suministrado la instrucción mediante un estado inicial.
- Un centro de control que a través de un número finito de estados se encarga de ejecutar el programa para el cual está destinada la máquina. Las instrucciones son llevadas a cabo por el apuntador y éste se dentendrá cuando algún estado especial se lo permita.

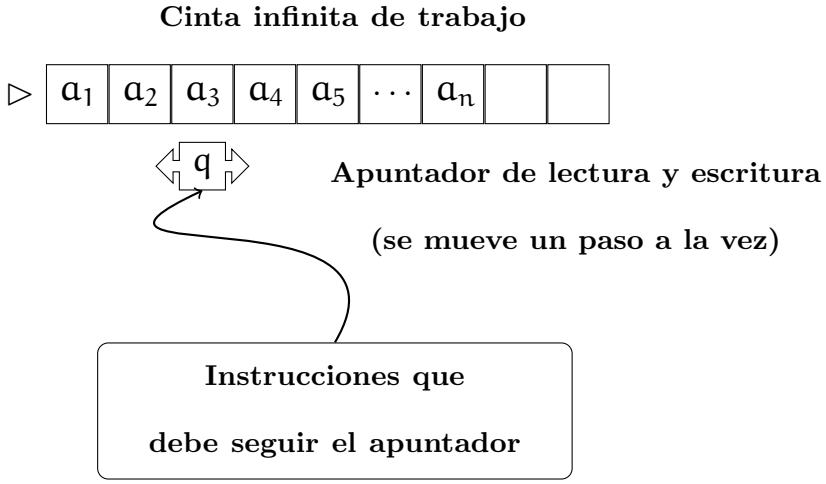

FIGURA 1.1. *Visualización de una Máquina de Turing.*

Formalmente:

DEFINICIÓN 1.13. Una *Máquina de Turing* es una terna $M = (K, \Sigma, \Delta)$ donde K es un conjunto finito no vacío de *estados* con un elemento especial s que llamamos *estado inicial*, Σ es un conjunto finito no vacío de símbolos que llamamos *alfabeto* (usualmente $\Sigma = \{0, 1\}$)

y Δ es un subconjunto de

$$\big(K \times (\Sigma \cup \{\sqcup\})\big) \times \big((K \cup \{q_a, q_n\}) \times (\Sigma \cup \{\sqcup\}) \times \{\leftarrow, \rightarrow, -\}\big).$$

Se asume que q_a (estado de aceptación), q_n (estado de negación), $\sqcup$ (símbolo vacío) y $\leftarrow, \rightarrow, -$ (direcciones del apuntador) no están en $K \cup \Sigma$. Decimos que M es *determinística* si existe una función $\delta : K \times (\Sigma \cup \{\sqcup\}) \to (K \cup \{q_a, q_n\}) \times (\Sigma \cup \{\sqcup\}) \times \{\leftarrow, \rightarrow, -\}$ tal que $\Delta = \{(q, a, \delta(q, a)) : (q, a) \in K \times (\Sigma \cup \{\sqcup\})\}$ y en este caso se suele denotar $M = (K, \Sigma, \delta)$. Si esto no sucede, decimos que M es *no determinística*. Una *entrada* de M es un elemento $x \in \bigcup_{n \in \mathbb{N}} \Sigma^n = \Sigma^*$. Si $x \in \Sigma^n$, definimos el *tamaño de* x como $|x| = n$.

La máquina M, como proceso algorítmico, se encarga de reconocer todos los pares (q, a) que aparecen a lo largo de las computaciones asociadas a la entrada x. Estas computaciones están registradas en el conjunto Δ. Por ejemplo, si $(q, a, p, b, \mathbf{d}) \in \Delta$, entonces el estado q cambiará por el estado $p \in K \cup \{q_a, q_n\}$, sobre el símbolo a se escribe el símbolo $b \in \Sigma \cup \{\sqcup\}$ y $\mathbf{d}$ indica cuál símbolo de x (o sobreescrito en x) será leído por el apuntador. En la siguiente figura se visualiza esta idea.

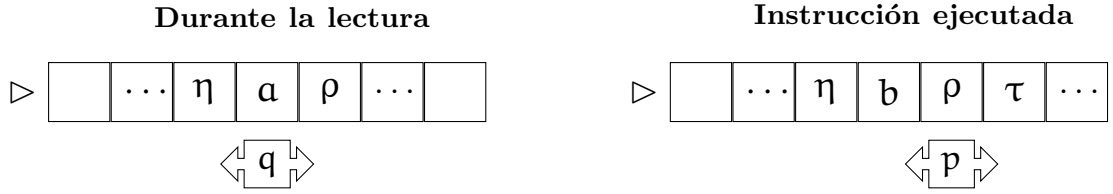

FIGURA 1.2. *Computación* $(q, a, p, b, \rightarrow)$ *ejecutada.*

Si $(p, \rho) \in K \times (\Sigma \cup \{\sqcup\})$ y existe $T \in (K \cup \{q_a, q_n\}) \times (\Sigma \cup \{\sqcup\}) \times \{\leftarrow, \rightarrow, -\}$ tal que $(p, \rho, T) \in \Delta$, se realiza un proceso similar al antes descrito.

Estamos principalmente interesados en aquellas máquinas M tales que para toda entrada x se realiza una cantidad finita de computaciones válidas hasta llegar a un estado de aceptación o de negación. En el caso de máquinas determinísticas, la condición de aceptación o negación se basa simplemente en recorrer todas y cada unas de las computaciones (q, a, T) que se ejecutan para la entrada x, desde el par inicial $(s, \triangleright)$, hasta llegar a un paso en el cual el siguiente estado sea q_a o q_n. Decimos que M *acepta a* x si el último estado es q_a y decimos que M *rechaza a* x si el último estado es q_n.

Para las máquinas no determinísticas, podrían existir pares $(q, a) \in K \times (\Sigma \cup \{\sqcup\})$ tales que $(q, a, T), (q, a, T') \in \Delta$ con $T \neq T'$. Una máquina no determinística se entiende como un procesador que ejecuta dos o más acciones paralelamente, es decir, la máquina debe ser capaz de realizar las acciones indicadas por T y T' a la vez, como se muestra en la FIGURA 1.3. La imagen muestra las distintas acciones que realiza una máquina cuando $(q, a, p, b, \rightarrow)$ y $(q, a, r, c, \leftarrow)$ son ambas computaciones posibles. Cada vez que esto ocurra, se producirá una *rama de decisión*, una por cada posible computación que la máquina no determinística M está capacitada a realizar. Una *hoja* de M es una tupla $(q, a, p, b, d) \in \Delta$ tal que p es el estado q_a o q_n.

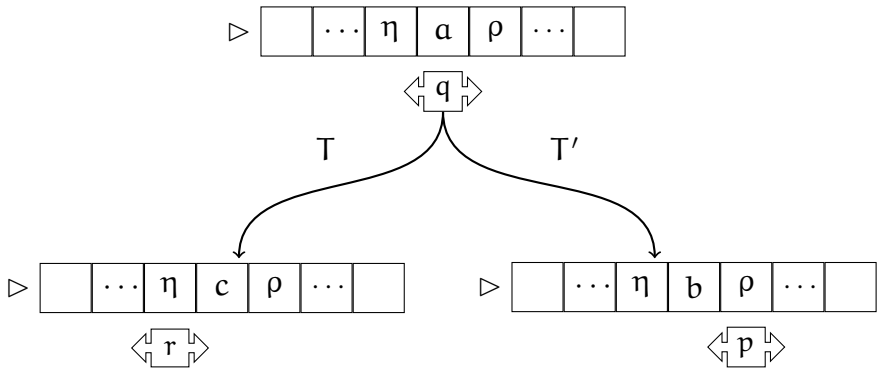

FIGURA 1.3. *Un par de computaciones ejecutadas no determinísticamente.*

La condición de aceptación para estas máquinas varía según el problema que se quiera resolver. Un tipo de aceptación, que llamaremos *de tipo I*, es el siguiente:

Suponga que M es tal que para toda entrada x sucede una de las dos:

- Existe una hoja que culmina en un estado de aceptación;
- Todas las hojas culminan en estado de negación.

Si x es tal que ocurre el primer caso, decimos que M acepta a x, de lo contrario, decimos que M rechaza a x. Note que este tipo de aceptación concuerda con otros conceptos que ya hemos visto como satisfabilidad y la clase $\exists\mathbf{SO}$.

Otro tipo de aceptación de interés, que llamaremos *de tipo II*, es el dual del tipo anterior. Suponga que M es tal que para toda entrada x sucede una de las dos:

- Todas las hojas culminan en estado de aceptación;
- Alguna hoja culmina en un estado de negación.

Si x es tal que ocurre el primer caso, decimos que M acepta a x, de lo contrario, decimos que M rechaza a x. Este tipo de aceptación concuerda con otros conceptos que ya hemos visto como insatisfabilidad, tautología y la clase $\forall\mathbf{SO}$. En el próximo capítulo veremos otros tipos de aceptación no determinísticas.

Definición 1.14. Sea M una máquina de Turing con alfabeto Σ. Un *lenguaje* es un conjunto $L \subseteq \Sigma^*$. Decimos que M *decide instancias en* L, o simplemente M *decide a* L, si

$$L = \{x \in \Sigma^* : M \text{ acepta a } x\}.$$

Si L representa un problema recursivo, escribimos $L = L(M)$ para especificar que M es la máquina que lo decide.

Las máquinas de Turing no sólo sirven como modelo computacional para solucionar problemas de decisión, también se pueden adaptar para simular computaciones asociadas a funciones. Antes de explicar este proceso requerimos una definición.

Definición 1.15. Una máquina de Turing $M = (K, \Sigma, \Delta)$ se dice que es *multicinta*[3] si Δ es un subconjunto de

$$\big(K \times (\Sigma \cup \{\sqcup\})^k\big) \times \big((K \cup \{q_a, q_n, q_h\}) \times (\Sigma \cup \{\sqcup\})^k \times \{\leftarrow, \rightarrow, -\}^k\big),$$

para algún $k \geq 2$. Se dice que M tiene una *cinta de lectura* si la primera componente de alfabeto de las tuplas $D \in \Delta$ sólo se utiliza para leer los símbolos de las entradas x y no para sobreescribir. Las siguientes $k-1$ componentes de alfabeto de las tuplas $D \in \Delta$ conforman las *cintas de trabajo*. Si la última componente sólo se utiliza para escribir símbolos se dice que M tiene una *cinta de salida*.

Todas las máquinas de Turing son adaptables a una con cinta de lectura, por lo que en adelante asumiremos que todas las máquinas son multicinta y con cinta de lectura. El nuevo estado q_h, denominado *estado de parada*, no se utiliza si M es una máquina de decisión. Si M no es una máquina de decisión, entonces M posee cinta de salida, es determinística y q_h representa el estado final de M en cualquier entrada x. Bajo estas condiciones, M define una función $f_M : \Sigma^* \to \Sigma^*$, donde $f_M(x)$ es la palabra escrita en la cinta de salida de M en

[3]Esta terminología surge por la idea intuitiva de Máquina de Turing, que nos permite visualizar este concepto como un mecanismo con múltiples componentes.

la entrada x, cuando ésta alcanza el estado q_h. Recíprocamente, toda máquina de Turing determinística se puede adaptar a una máquina con cinta de salida.

Este tipo de máquinas que definen funciones son también relevantes en el estudio de máquinas no determinísticas. Antes de explicar esto, necesitamos definir la *eficiencia* de una máquina de Turing. El desarrollo que mostramos a continuación lo adoptamos de [2].

DEFINICIÓN 1.16. Sean $M = (K, \Sigma, \delta)$ una máquina de Turing, $f : \Sigma^* \to \Sigma^*$ y $T : \mathbb{N} \to \mathbb{N}$ un par de funciones. Decimos que M *computa* f si $f = f_M$. Decimos que M *computa* f *en tiempo* $T(n)$ si $f = f_M$ y M realiza a lo sumo $T(|x|)$ computaciones para obtener $f(x)$.

En este último planteamiento se define en realidad una cota superior sobre el número de operaciones válidas realizadas por una máquina de Turing. Como las máquinas que estamos considerando realizan finitas computaciones en todas sus entradas x, la función T siempre se puede definir con exactitud. Para toda entrada x existen secuencias $(C_1, \ldots, C_m)$ de computaciones válidas asociadas a x, donde C_1 y C_m son, respectivamente, la primera y última computación realizadas por M en la entrada x. Para todo número natural n, se define $T(n)$ como el máximo valor m sobre todas las entradas x de tamaño n. Esta función T no sólo es compatible con la DEFINICIÓN 1.16, sino que es la mínima cota a considerar. De esta manera podemos hablar del *tiempo* (exacto) que tarda una máquina de Turing en dar una respuesta y siempre podremos valernos de este parámetro.

Otro tipo de recurso que nos interesa es el *espacio* utilizado por una máquina de Turing. Recordemos que todas las máquinas a considerar tienen una cinta de entrada y cintas de trabajo, las cuales se encuentran vacías en el instante inicial. Cuando una entrada se suministra y la máquina comienza a operar, las cintas de trabajo empiezan a almacenar información. El número de celdas utilizadas o requeridas por M para realizar sus computaciones es el espacio utilizado por M. Formalmente,

DEFINICIÓN 1.17. Sean $M = (K, \Sigma, \delta)$ una máquina de Turing, $f : \Sigma^* \to \Sigma^*$ y $S : \mathbb{N} \to \mathbb{N}$ un par de funciones. Decimos que M *computa* f *con espacio* $S(n)$ si $f = f_M$ y existe $c \in \mathbb{N}$ tal que M utiliza a lo sumo $c \cdot S(|x|)$ celdas en sus cintas de trabajo para obtener $f(x)$.

Una vez más, la definición anterior se refiere a una cota superior sobre el espacio utilizado por la máquina, pero ésta se puede definir con exactitud mediante un proceso similar al ya

descrito para el parámetro tiempo. En la siguiente definición se introducen el tipo de cotas que nos interesan.

Definición 1.18. Sea el alfabeto $\Sigma = \{0, 1\}$ y una función $T : \mathbb{N} \to \mathbb{N}$. Decimos que T es una *función de complejidad propia* si existe una máquina de Turing determinística M con alfabeto Σ tal que, al darle la entrada 1^n (la palabra de tamaño n compuesta sólo de 1's), M escribe $f(n)$ en binario, usando tiempo $T(n)$ y espacio $S(n) = \mathcal{O}(T(n))$.[4]

Si bien esta definición parece bastante restrictiva, en realidad engloba una gran cantidad de funciones conocidas como $\lceil \log(n) \rceil$, n, n^k, 2^n, etc; y deja por fuera funciones patológicas que son computacionalmente difíciles de describir. En adelante, asumiremos que todas las funciones mencionadas son de complejidad propia. Hasta ahora sólo nos hemos limitado a describir los recursos utilizados por máquinas determinísticas, justamente por la relación que éstas tienen con las máquinas que emulan funciones. En el caso de máquinas no determinísticas, los parámetros tiempo y espacio se calculan considerando cada rama de decisión como un proceso determinístico y tomando el máximo sobre todos los valores obtenidos. En las siguientes definiciones, cuando nos refiramos a una máquina no determinística, asumimos que su modo de aceptación es de tipo I.

Definición 1.19. Una *clase de complejidad* es un conjunto de lenguajes que pueden ser decididos por máquinas de Turing cuya efectividad temporal o espacial se encuentra acotada por una función de complejidad propia. Sea $T(n)$ una función de complejidad propia. Se definen las siguientes clases de complejidad:

- **TIME**$[T(n)]$ es la clase de los lenguajes decidibles por máquinas de Turing determinísticas que operan en tiempo $\mathcal{O}(T(n))$;
- **NTIME**$[T(n)]$ es la clase de los lenguajes decidibles por máquinas de Turing no determinísticas que operan en tiempo $\mathcal{O}(T(n))$;
- **SPACE**$[T(n)]$ es la clase de los lenguajes decidibles por máquinas de Turing determinísticas que utilizan espacio $\mathcal{O}(T(n))$;
- **NSPACE**$[T(n)]$ es la clase de los lenguajes decidibles por máquinas de Turing no determinísticas que utilizan espacio $\mathcal{O}(T(n))$.

[4] Informalmente, $\mathcal{O}$ es una relación de proporcionalidad que indica que S crece tanto como T.

Decimos que una clase de complejidad es determinística o no determinística según el tipo de máquinas que se utilicen. Varios aspectos de la computabilidad, básicos y otros más profundos, permiten relacionar estas clases entre sí. En [21] se consigue una demostración detallada de las siguientes relaciones:

TEOREMA 1.3. Si $T(n)$ es una función de complejidad propia, entonces:

1. $\mathbf{SPACE}[T(n)] \subseteq \mathbf{NSPACE}[T(n)]$ y $\mathbf{TIME}[T(n)] \subseteq \mathbf{NTIME}[T(n)]$.
2. $\mathbf{NTIME}[T(n)] \subseteq \mathbf{SPACE}[T(n)]$.
3. $\mathbf{NSPACE}[T(n)] \subseteq \mathbf{TIME}\left[2^{\lceil \log(n) \rceil + T(n)}\right]$.

Algunas de las clases de complejidad más estudiadas son:

- $\mathbf{L} = \mathbf{SPACE}[\log(n)]$;
- $\mathbf{NL} = \mathbf{NSPACE}[\log(n)]$;
- $\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}[n^k]$;
- $\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathbf{NTIME}[n^k]$;
- $\mathbf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathbf{SPACE}[n^k]$;
- $\mathbf{EXPTIME} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}\left[2^{n^k}\right]$.

Por el TEOREMA 1.3 se obtiene la siguiente cadena de contenciones:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}.$$

Se sabe que $\mathbf{L} \neq \mathbf{PSPACE}$ y por lo tanto alguna de estas contenciones debe ser necesariamente propia. De esta cadena se desglosan varios de los problemas centrales de la complejidad computacional, incluyendo el ya mencionado **P versus NP**.

DEFINICIÓN 1.20. Sea $\mathbf{C}$ una clase de complejidad. Definimos el *complemento* de $\mathbf{C}$ como la clase de lenguajes $\mathbf{coC} = \{L : L^c \in \mathbf{C}\}$. [5]

En términos de eficiencia temporal o espacial, un lenguaje es tan difícil de decidir como su complemento, lo que puede variar es el tipo de aceptación con que se decide el lenguaje. Si $\mathbf{C}$ es una clase de complejidad determinística, entonces $\mathbf{C} = \mathbf{coC}$, puesto que el tipo de aceptación en $\mathbf{C}$ es el mismo que en $\mathbf{coC}$. Si $\mathbf{C}$ es no determinística, no hay una conclusión evidente que relacione a esta clase con su complemento, al menos no en un contexto general como en el caso determinístico. En [15] se demuestra que para cualquier función de

[5] L^c es el complemento usual: si $L \subseteq \Sigma^*$, para algún vocabulario Σ, entonces $L^c = \Sigma^* - L$.

complejidad propia $S(n) \geq \log(n)$, $\mathbf{NSPACE}[S(n)] = \mathbf{coNSPACE}[S(n)]$. Por otro lado, hasta ahora no se sabe si **NP** y **coNP** son iguales. Una respuesta afirmativa a la conjetura $\mathbf{NP} \neq \mathbf{coNP}$ implicaría que $\mathbf{P} \neq \mathbf{NP}$.

Ahora, las clases **coC** son efectivamente clases de complejidad, lo cual queda claro cuando **C** es determinística. Si **C** es no determinística, un lenguaje $L \in \mathbf{coC}$ se puede decidir con una máquina no determinística con tipo de aceptación II. Otra caracterización de la clase **NP**, que nos interesará generalizar más adelante, es la siguiente:

PROPOSICIÓN 1.4. Un lenguaje $L \subseteq \{0,1\}^*$ está en **NP** si y sólo si existe un polinomio $p(n)$ y una máquina determinística M que opera en tiempo polinomial, tales que para todo $x \in \{0,1\}^*$, se cumple que

$$x \in L \quad \Leftrightarrow \quad \exists u \in \{0,1\}^{p(|x|)} \text{ tal que } M \text{ acepta a } xu,$$

donde xu es la concatenación de la entrada x con la palabra u. Si $x \in L$, a u se le llama *certificado*, o *testigo*, de x.

En la siguiente sección mostraremos a través de propiedades sintácticas, la inclusión de varios problemas de decisión en algunas de las clases de complejidad ya vistas.

1.5. Complejidad Descriptiva

La Teoría Descriptiva de la Computabilidad estudia la Complejidad Computacional desde la perspectiva de la Lógica. El teorema de Fagin, que establece una relación entre el fragmento de sentencias de segundo orden $\exists\mathbf{SO}$ y la clase de complejidad **NP**, fue el primer resultado en este ámbito.

No hemos aclarado en qué sentido los lenguajes decidibles por máquinas de Turing representan problemas de decisión, y mucho menos por qué el alfabeto $\Sigma = \{0,1\}$ parece bastar como vocabulario de las máquinas en general. Para responder cada una de estas preguntas, consideremos la siguiente:

DEFINICIÓN 1.21. Sean σ y τ vocabularios relacionales, con $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$, y sea k un número entero positivo. Definiremos una aplicación $I : \mathrm{STRUC}[\tau] \to \mathrm{STRUC}[\sigma]$ dada por una tupla $\varphi_0, \varphi_1, \ldots, \varphi_r, \psi_1, \ldots, \psi_s$ de $r + s + 1$ fórmulas de primer orden sobre

τ. A cada estructura $\mathcal{A} \in \text{STRUC}[\tau]$ se le asocia una estructura $I(\mathcal{A}) \in \text{STRUC}[\sigma]$,

$$I(\mathcal{A}) = \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_s^{I(\mathcal{A})} \rangle,$$

donde:

- El universo $|I(\mathcal{A})|$ es un subconjunto $|\mathcal{A}|^k$ dado por

$$|I(\mathcal{A})| = \{(b_1, \ldots, b_k) \in |\mathcal{A}|^k : \mathcal{A} \models \varphi_0(b_1, \ldots, b_k)\},$$

  y φ_0 es una fórmula numérica.
- Cada relación $R_i^{I(\mathcal{A})}$ es un subconjunto de $|I(\mathcal{A})|^{a_i}$ definible en primer orden por φ_i:

$$R_i^{I(\mathcal{A})} = \left\{((b_1^1, \ldots, b_1^k), \ldots, (b_{a_i}^1, \ldots, b_{a_i}^k)) \in |I(\mathcal{A})|^{a_i} : \mathcal{A} \vDash \varphi_i(b_1^1, \ldots, b_{a_i}^k)\right\}.$$

- Cada constante $c_j^{I(\mathcal{A})}$ es un elemento de $I(\mathcal{A})$ definible en primer orden por ψ_j: se identifica a $c_j^{I(\mathcal{A})}$ con la única tupla $(b_1, \ldots, b_k) \in I(\mathcal{A})$ tal que $\mathcal{A} \vDash \psi_j(b_1, \ldots, b_k)$.

Denotamos $I = \lambda_{x_1^1 \ldots x_a^k}\langle \varphi_0, \varphi_1, \ldots, \psi_s \rangle$, donde $a = \text{máx}\{a_1, \ldots, a_r\}$, y decimos que I es una *interpretación de primer orden de aridad* k de $\text{STRUC}[\tau]$ a $\text{STRUC}[\sigma]$. Cuando $|I(\mathcal{A})|$ se defina como $|\mathcal{A}|^k$, tomamos la expresión $\varphi_0 \equiv \mathbf{true}$.

Las interpretaciones de primer orden son el medio que nos permitirán movernos entre problemas y representan además el análogo en teoría descriptiva de reducciones computacionales. Como un primer ejemplo de esto, veamos cómo codificar en binario una estructura.

Sea el vocabulario $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ y $\mathcal{A} \in \text{STRUC}[\sigma]$ cuyo dominio $|\mathcal{A}|$ es el conjunto $[n] = \{0, 1, \ldots, n-1\}$. Consideremos el orden natural de $[n]$, el cual induce el orden lexicográfico sobre los conjuntos $|\mathcal{A}|^m$ para todo natural m. Nos valemos de este orden para codificar en binario a las relaciones $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$. Para cada $i = 1, \ldots, r$ definamos $\text{bin}(R_i^{\mathcal{A}})$ como la palabra binaria de tamaño n^{a_i} donde un "1" en una posición dada indica que la correspondiente tupla en $|\mathcal{A}|^{a_i}$ con el orden lexicográfico está en $R_i^{\mathcal{A}}$. Para las constantes $c_j^{\mathcal{A}} \in [n]$ con $j = 1, \ldots, s$ se define $\text{bin}(c_j^{\mathcal{A}})$ como la palabra en $\lceil \log(n) \rceil$ bits que codifica a $c_j^{\mathcal{A}}$ en binario. $\text{bin}(c_j^{\mathcal{A}})$ tiene tantos ceros a la izquierda como sean necesarios. La *codificación binaria* de la estructura $\mathcal{A}$ la definimos como la palabra $\text{bin}_\sigma(\mathcal{A})$ dada por la concatenación de las relaciones y constantes codificadas, es decir,

$$\text{bin}_\sigma(\mathcal{A}) = \text{bin}(R_1^{\mathcal{A}}) \cdots \text{bin}(R_r^{\mathcal{A}})\, \text{bin}(c_1^{\mathcal{A}}) \cdots \text{bin}(c_s^{\mathcal{A}}). \tag{1.4}$$

Es de notar que el tamaño de $\text{bin}_\sigma(\mathcal{A})$ es

$$\widehat{n}_\sigma = n^{a_1} + \cdots + n^{a_r} + s\lceil \log(n) \rceil. \tag{1.5}$$

Cuando el vocabulario se sobreentienda no hará falta el subíndice σ, por lo que será usual denotar a las codificaciones simplemente por $\text{bin}(\mathcal{A})$ y a su longitud por $\widehat{n}$. En la siguiente proposición se demostrará que las codificaciones binarias se pueden obtener mediante una interpretación de primer orden.

PROPOSICIÓN 1.5. Sean los vocabularios $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ y $\sigma_s = \langle S^1 \rangle$. La función bin_σ es descrita por una interpretación de primer orden $\beta_\sigma : \text{STRUC}[\sigma] \to \text{STRUC}[\sigma_s]$.

DEMOSTRACIÓN. Definiremos β_σ explícitamente. Primero necesitamos encontrar un valor k lo suficientemente grande para cubrir la cantidad $\widehat{n}$ de bits requeridos para la codificación binaria (1.4). Sea $a = \max\{a_1, \ldots, a_r\}$. El siguiente valor resultará adecuado:

$$k = r + a + s + 1.$$

Dada una σ-estructura $\mathcal{A}$ con dominio $[n]$, la interpretación $\mathcal{B} = \beta_\sigma(\mathcal{A})$ tendrá en el universo elementos de la forma

$$\overline{x} = (x_1, \ldots, x_r, y_1, \ldots, y_a, z_1, \ldots, z_s, w) \in |\mathcal{A}|^k,$$

donde las variables x_i, y_j, z_m y w estarán sujetas a ciertas condiciones dadas por una fórmula de primer orden numérica φ_0 que describiremos a continuación. $\varphi_0(\overline{x})$ será la conjunción de las siguientes fórmulas:

- $(x_i = 0 \vee x_i = 1)$ para todo $i = 1, \ldots, r$;
- $(z_m = 0 \vee z_m = 1)$ para todo $m = 1, \ldots, s$;
- Si $x_i = 1$ entonces $x_j = 0$ para todo $j \neq i$ y $z_m = 0$ para todo $m = 1, \ldots, s$;
- Si $z_m = 1$ entonces $z_j = 0$ para todo $j \neq m$ y $x_i = 0$ para todo $i = 1, \ldots, r$;
- Si $x_i = 1$ entonces $y_j = 0$ para todo $j = a_i + 1, \ldots, a$ y $w = 0$;
- Si $z_m = 1$ entonces $y_j = 0$ para todo $j = 1, \ldots, a$ y $0 \leq w \leq \lceil \log(\max) \rceil$.

Las fórmulas de los cuatros primeros ítems indican que las variables x_i y z_m se usan para referirse a una y sólo una relación o constante a la vez. Si $x_i = 1$ para algún i, las variables $y_1, \ldots, y_{a_i}$ son libres, y son estas tuplas las que nos permitirán codificar a la relación $R_i^{\mathcal{A}}$. Si $z_m = 1$ para algún m, la variable w nos indicará el bit de la constante $c_m^{\mathcal{A}}$ por analizar. Note

que φ_0 es en efecto numérica, pues ninguna de estas subfórmulas depende de los símbolos relacionales de σ y además el conjunto

$$|\mathcal{B}| = |\beta_\sigma(\mathcal{A})| = \{\overline{x} \in |\mathcal{A}|^k : \mathcal{A} \vDash \varphi_0(\overline{x})\}$$

tiene exactamente $\widehat{n}$ elementos (1.5). Necesitamos ahora incluir a ciertos elementos de $|\mathcal{B}|$ en un subconjunto S de manera tal que $\langle |\mathcal{B}|, S^{\mathcal{B}}\rangle$ sea justamente la codificación $\mathrm{bin}(\mathcal{A})$. Haremos esto a través de una fórmula de primer orden φ_S que describiremos a continuación. $\varphi_S(\overline{x})$ será la disyunción de las siguientes fórmulas:

- $(z_m = 1) \wedge \mathrm{bit}(c_m, w)$ con $m = 1, \ldots, s$;
- $(x_i = 1) \wedge R_i(y_1, \ldots, y_{a_i})$ con $i = 1, \ldots, r$.

Estas fórmulas ponen en total correspondencia a los elementos de $|\mathcal{B}|$ con cada una de los bits de la codificación binaria $\mathrm{bin}(\mathcal{A})$. Más aún, considerando el orden lexicográfico restricto a $|\mathcal{B}|$ se obtiene que $\mathrm{bin}(\mathcal{A})$, como σ_s-estructura, es exactamente $\langle |\mathcal{B}|, S^{\mathcal{B}}\rangle$. A la interpretación β_σ la denotamos por $\lambda_{\overline{x}}\langle \varphi_0, \varphi_S\rangle$. □

Bajo todas las condiciones establecidas, β_σ es una función inyectiva. Esto implica que dada una σ_s-estructura de la forma $\beta_\sigma(\mathcal{A})$, siempre será posible obtener la σ-estructura $\mathcal{A}$ explícitamente. Más aún, esta decodificación también se puede hacer por medio de una interpretación de primer orden.

Ya queda claro entonces en qué sentido los problemas de decisión pueden decidirse por máquinas de Turing, pues una vez escogido el vocabulario con el cual interpretar instancias sólo nos queda reinterpretarlas en binario, es decir, sobre el alfabeto $\Sigma = \{0, 1\}$. Una máquina de Turing puede verificar en espacio logarítmico si el tamaño de una entrada corresponde a un valor $\widehat{n}$ y luego realizar el resto de sus computaciones para decidir instancia sobre el problema en cuestión.

Todos los problemas de decisión los denotaremos con un nombre en letras mayúsculas. Por ejemplo, el problema de satisfabilidad para fórmulas en FNC lo denotamos por SAT, insatisfabilidad por UNSAT y tautología para fórmulas en FND por VALID. ¿Qué sucede cuando un problema es caracterizable por un lenguaje lógico como **FO** o **SO**? Recordemos que según los Ejemplos 1.4 y 1.5 algunos problemas relacionados con la lógica Booleana son más difíciles de responder según las instancias que se consideren y esto se corresponde incluso con los tipos de lenguajes lógicos con que se expresen.

Sea σ un vocabulario y $\Phi \in \mathbf{SO}[\sigma]$ una sentencia. Definimos el conjunto de *modelos* de Φ como $\mathrm{MOD}[\Phi] = \{\mathcal{A} \in \mathrm{STRUC}[\sigma] : \mathcal{A} \models \Phi\}$. Dada una σ-estructura $\mathcal{A}$ cabe preguntarse entonces si $\mathcal{A} \in \mathrm{MOD}[\Phi]$. En otras palabras, toda sentencia Φ de segundo orden define un problema de decisión. De esta manera, identificamos

$$\mathrm{SAT} = \mathrm{MOD}[\Phi_{\mathrm{sat}}], \qquad \mathrm{UNSAT} = \mathrm{MOD}[\Phi_{\mathrm{unsat}}], \qquad \mathrm{VALID} = \mathrm{MOD}[\Phi_{\mathrm{tauto}}]$$

donde Φ_{sat}, Φ_{unsat} y Φ_{tauto} son las fórmulas de segundo orden planteadas en el Ejemplo 1.5. Si Φ se restringe a sentencias de primer orden, la dificultad del problema $\mathrm{MOD}[\Phi]$ es baja, como se muestra en [3].

Proposición 1.6. $\mathrm{MOD}[\varphi]$, con $\varphi \in \mathbf{FO}$, pertenece a la clase de complejidad $\mathbf{L}$.

Esta proposición sugiere que la codificación binaria de una estructura, o cualquier otra operación expresable en primer orden, es eficiente, pues no requiere de muchos recursos computacionales para poderse ejecutar.

En adelante, cuando nos refiramos a una lógica $\mathcal{L}$, ésta puede tratarse de **FO**, **SO**, o cualquier fragmento cuantificado de ambas, como $\exists\mathbf{SO}$ o $\forall\mathbf{SO}$.

Definición 1.22. Sea $\mathcal{L}$ una lógica y $\mathbf{C}$ una clase de complejidad. Decimos que $\mathcal{L}$ *captura a* $\mathbf{C}$ si se cumplen las siguientes condiciones:

1. Todo problema $\mathrm{MOD}[\Phi]$, con $\Phi \in \mathcal{L}$, es decidible en $\mathbf{C}$;
2. Para todo problema $\mathrm{L} \in \mathbf{C}$, existe $\Phi \in \mathcal{L}$ tal que $\mathrm{L} = \mathrm{MOD}[\Phi]$.

Si este es el caso, escribimos $\mathbf{C} = \mathcal{L}$.[6]

Como un primer ejemplo de esta definición presentamos el *Teorema de Fagin*:

Teorema 1.7. $\mathbf{NP} = \exists\mathbf{SO}$.

La Proposición 1.6 se puede reescribir como $\mathbf{FO} \subseteq \mathbf{L}$, pero esta contención es estricta [3]. Los recursos expresivos de **FO** no son lo suficientemente fuertes para capturar a la clase $\mathbf{L}$. La lógica **SO**, por otra parte, tiene demasiado poder expresivo, sólo un fragmento de **SO** permite capturar en su totalidad a la clase **NP**. En este trabajo estamos especialmente interesados en clases de mayor complejidad que **NP**, por lo que no repasaremos teoría descriptiva para las clases $\mathbf{L}$, $\mathbf{NL}$ o $\mathbf{P}$. Citamos el artículo [12], donde estas clases se caracterizan.

[6] Hay un abuso en esta notación, pero la mayoría de los libros dedicados al tema la adoptan.

El Teorema 1.7 ofrece un método, relativamente sencillo, para verificar si un problema pertenece a **NP** sin tener que recurrir a formalismos computacionales. Por ejemplo, el problema SAT está en **NP**, pues $\text{SAT} = \text{MOD}[\Phi_{\text{sat}}]$ y $\Phi_{\text{sat}} \in \exists\mathbf{SO}$. Como consecuencia directa del Teorema de Fagin tenemos

Corolario 1.8. $\mathbf{coNP} = \forall\mathbf{SO}$.

De esta manera obtenemos que UNSAT y VALID están en **coNP**. A lo largo del trabajo nos valemos de propiedades similares para clasificar ciertos problemas. Hemos visto distintas equivalencias relacionadas con **NP**. Otra manera de estudiar esta clase es a través de problemas particulares. La siguiente definición es fundamental en el estudio de la computabilidad:

Definición 1.23. Sean los lenguajes $L, L' \subseteq \{0,1\}^*$. Una *reducción* de L a L' es una función $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$, $x \in L$ si y sólo si $f(x) \in L'$. Si la reducción f es computable por una máquina de Turing determinística que opera en tiempo polinomial sobre la longitud de la entrada $x \in \{0,1\}^*$, se dice que L es *reducible polinomialmente a* L' y denotamos $L \leq_{\text{poly}} L'$. Si f es computable por una máquina de Turing determinística que utiliza espacio logarítmico sobre la longitud de la entrada $x \in \{0,1\}^*$, se dice que L es *reducible en espacio logarítmico a* L' y denotamos $L \leq_{\log} L'$.

Tanto la relación $\leq_{\text{poly}}$ como $\leq_{\log}$ son transitivas. El primer resultado importante en teoría de reducibilidad se conoce como *Teorema de Cook* [8] y plantea lo siguiente:

Teorema 1.9. SAT es **NP**-*completo* vía reducciones polinomiales, es decir, todo problema en **NP** se puede reducir a SAT polinomialmente.

Las reducciones polinomiales son adecuadas para definir *completitud* en clases de mayor complejidad que **NP**. Para clases de menor complejidad como **L**, **NL** o **P** no resulta viable utilizar una cantidad tal de recursos temporales para reducir un problema a otro, por lo que se suelen utilizar reducciones logarítmicas. De ser cierta la contención propia $\mathbf{L} \subsetneq \mathbf{P}$, las reducciones logarítmicas resultarían, teóricamente, más eficientes que las reducciones polinomiales. Podemos valernos de la Definición 1.21 para definir reducibilidad en términos de la lógica de primer orden.

Definición 1.24. Sean τ y σ vocabularios, y sean $L \subseteq \text{STRUC}[\tau]$ y $L' \subseteq \text{STRUC}[\sigma]$ un par de problemas. Decimos que L es *reducible en primer orden a* L' si existe una interpretación

I de STRUC[τ] a STRUC[σ] tal que para toda instancia $\mathcal{A}$ de L se cumple que

$$\mathcal{A} \in \mathsf{L} \quad \Leftrightarrow \quad \mathrm{I}(\mathcal{A}) \in \mathsf{L}',$$

y denotamos esto por $\mathsf{L} \leq_{\mathsf{fo}} \mathsf{L}'$. A I la llamamos *reducción de primer orden.*

No se conoce ningún problema que sea completo en su respectiva clase a través de reducciones polinomiales o logarítmicas y que no lo sea vía reducciones de primer orden. Lo sorprendente de este aspecto es que, por ser cierto **FO** $\subset$ **L**, la mayor dificultad de reducir un problema a otro es justamente definir la reducción de primer orden I, pues una vez conseguida ésta, no se requieren de muchos recursos computacionales para ejecutarla. Siguiendo en esta línea de investigación, Immerman y Landau probaron varias propiedades básicas de un tipo especial de reducciones de primer orden [14]:

Definición 1.25. Sean L y L′ como en la definición anterior y supongamos que $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. Decimos que la reducción $\mathrm{I} = \lambda_{x_1^1 \ldots x_a^k} \langle \varphi_0, \varphi_1, \ldots, \psi_s \rangle$ es una *proyección* si cada fórmula φ_i y ψ_j satisface la condición de proyección, es decir, cada una de estas fórmulas es equivalente a una como

$$\alpha_1 \vee (\alpha_2 \wedge \lambda_2) \vee \cdots \vee (\alpha_\ell \wedge \lambda_\ell), \tag{1.6}$$

denominada *fórmula proyectiva*, y tal que las subfórmulas α son fórmulas numéricas mutuamente excluyentes y las subfórmulas λ son literales no numéricos del vocabulario τ. Si este es el caso, denotamos $\mathsf{L} \leq_{\mathsf{fop}} \mathsf{L}'$.

Tanto la relación $\leq_{\mathsf{fo}}$ como $\leq_{\mathsf{fop}}$ son transitivas. Un par de resultados destacados que aparecen en [18] son:

Proposición 1.10. SAT es **NP**-completo vía proyecciones.

Proposición 1.11. Existe un problema **NP**-completo vía reducciones de primer orden, pero no vía proyecciones.

A pesar de que la condición de proyección parece ser muy restrictiva, este tipo de fórmulas son en realidad bastante recurrentes. Por ejemplo, $\varphi_S(\overline{x})$ en la Demostración 1.5 es siempre una fórmula proyectiva, sin importar cómo sea el vocabulario σ. En adelante, cuando nos refiramos a un problema **C**-completo, se sobreentiende que es vía proyecciones. En caso de ser completo con respecto a otro tipo de reducción, se especificará.

Ejemplo 1.6. Una consecuencia directa de la Proposición 1.10 es que UNSAT es **coNP**-completo. Valiéndonos de la transitividad de $\leq_{\mathsf{fop}}$ y de las propiedades conocidas de fórmulas Booleanas, demostraremos que VALID es también **coNP**-completo. Debemos construir una proyección $\mathrm{I} : \mathrm{STRUC}[\sigma_{\mathrm{fnc}}] \to \mathrm{STRUC}[\sigma_{\mathrm{fnd}}]$ tal que para toda instancia $\mathcal{A} \in \mathrm{STRUC}[\sigma_{\mathrm{fnc}}]$ se cumple que

$$\mathcal{A} \in \mathrm{UNSAT} \quad \Leftrightarrow \quad \mathrm{I}(\mathcal{A}) \in \mathrm{VALID}\,.$$

Definamos I como una simple descripción de las leyes de De Morgan: sea $\mathcal{A} = \langle [n], P^{\mathcal{A}}, N^{\mathcal{A}} \rangle$. La negación de la fórmula Booleana $\mathcal{A}$ es $\mathrm{I}(\mathcal{A}) = \langle [n], Q^{\mathrm{I}(\mathcal{A})}, M^{\mathrm{I}(\mathcal{A})} \rangle$, donde $Q^{\mathrm{I}(\mathcal{A})} = N^{\mathcal{A}}$ y $M^{\mathrm{I}(\mathcal{A})} = P^{\mathcal{A}}$. Entonces, $\mathrm{I} = \lambda_{xy}\langle \mathbf{true}, \varphi_Q, \varphi_M \rangle$ de aridad 1, con:

$$\varphi_Q(x, y) \equiv N(x, y) \qquad \text{y} \qquad \varphi_M(x, y) \equiv P(x, y),$$

las cuales cumplen con la condición de proyección. □

Hasta aquí hemos ejemplificado el modo de proceder a lo largo del trabajo. Primero introduciremos clases de complejidad **C** mediante procesos computacionales, y a los cuales se les asociarán lenguajes formales que los capturen. Luego, se propondrán problemas en base a esta nueva caracterización, y finalmente, se demostrará su completitud utilizando distintos métodos.

Capítulo 2

La Clase Σ_2^p

2.1. La Jerarquía Polinomial

En 1976, Larry Stockmeyer define la clase de complejidad **PH** (notación para Polynomial-Time Hierarchy) a través de Máquinas de Turing con Oráculo [24], un tipo especial de Máquinas de Turing que generalizan los procesos computacionales de las ya conocidas clases **P** y **NP**. Intuitivamente, un oráculo es un dispositivo abstracto que le permite a una máquina decidir o responder una pregunta en un sólo paso, lo cual minimiza considerablemente la cantidad de recursos temporales utilizados.

Definición 2.1. Sea el alfabeto $\Sigma = \{0, 1\}$. Una *máquina de Turing con oráculo* es una máquina de Turing multicinta $M^? = (K, \Sigma, \Delta)$ que cuenta con una cinta especial denominada *cinta de consulta* y posee también tres estados especiales $q_c, q_{sí}, q_{no} \in K$. El modo de proceder de $M^?$ es muy similar al de las máquinas ordinarias salvo que antes de proporcionarle una entrada x se debe especificar un conjunto $O \subseteq \Sigma^*$ que servirá de *oráculo*.

M^O actúa de la siguiente manera: dada una entrada x, M^O realiza los computos como lo haría cualquier máquina de Turing, proporcionando después de una cantidad finita de pasos una palabra $y \in \Sigma^*$ escrita en la cinta de consulta. A esta máquina le interesa saber si y está en O para poder continuar con sus cómputos y es en ese momento cuando los estados especiales entran en acción. El estado q_c nos indica el momento en el que para la máquina M^O es indispensable dar una respuesta a la pregunta "¿$y \in O$?". En el siguiente paso se obtendrá un cambio de estado como respuesta a esta pregunta: $q_{sí}$ o q_{no}, dependiendo del caso. Según la respuesta que se obtenga M^O continúa con sus cómputos hasta dar una respuesta de aceptación o de negación para x.

La eficiencia temporal de $M^?$ se mide como la de cualquier otra máquina no determinística. Para la eficiencia espacial no se considera el espacio utilizado en la cinta de consulta, pues ésta suele ser polinomial sobre el tamaño de la entrada x.

DEFINICIÓN 2.2. Sea $\Sigma_0^p = \mathbf{P}$ el nivel 0 de la *Jerarquía Polinomial*. Los niveles superiores de la Jerarquía se definen inductivamente: para $k \geq 0$, sea

$$\Sigma_{k+1}^p = \mathbf{NP}^{\Sigma_k^p} = \bigcup_{B \in \Sigma_k^p} \mathbf{NP}^B,$$

donde,

$$\mathbf{NP}^B = \{L\left(M^B\right) : \quad M \text{ es una máquina de Turing no determinística con oráculo } B, \text{ que opera en tiempo polinomial}\}.$$

De esta manera, se define $\mathbf{PH} = \bigcup_{k \geq 0} \Sigma_k^p$. También son de principal interés las clases

$$\Pi_{k+1}^p = \mathbf{coNP}^{\Sigma_k^p} = \mathbf{co}\Sigma_{k+1}^p.^{1}$$

Intuitivamente, para $k \geq 0$, la clase Σ_{k+1}^p contiene aquellos problemas que serían resolubles en $\mathbf{NP}$ con ayuda de un oráculo en Σ_k^p. Para mostrar a través de procesos computacionales que un problema está en alguno de estos niveles resulta más conveniente usar *certificados*. En nuestro caso tampoco nos valemos directamente de este método, pero nos servirá para deducir una caracterización sintáctica de Σ_2^p.

DEFINICIÓN 2.3. Sea Σ un alfabeto y sea $R \subseteq \Sigma^* \times \Sigma^*$. Decimos que R es *polinomialmente balanceado* si $(x, y) \in R$ implica que $|y| \leq |x|^a$ para algún entero $a \geq 1$ fijo.

El siguiente teorema es una generalización de la PROPOSICIÓN 1.4. éste y las subsiguientes proposiciones se suelen proponer como ejercicios en los textos dedicados al tema.

TEOREMA 2.1. Sea Σ un alfabeto y $L \subseteq \Sigma^*$ un lenguaje. $L \in \Sigma_{k+1}^p$ con $k \geq 1$ si y sólo si existe una relación R polinomialmente balanceada, decidible en Π_k^p y tal que

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

DEMOSTRACIÓN. Demostraremos el caso $k = 1$. Supongamos primero que tal relación R existe y describamos una máquina $M^?$ que decida a L en Σ_2^p. Sea x una instancia de L. Para saber si $x \in L$, la máquina M copia a x en una cinta especial (la cita de consulta) y seguido procede a construir no determinísticamente una palabra y, cuyo tamaño estará acotado superiormente por $|x|^a$, con a entero positivo fijo. Una vez escrita la palabra xy

[1] La segunda igualdad se obtiene por dualidad con el tipo de aceptación.

en la cinta especial, haremos uso de un oráculo adecuado para decidir L. Como $R \in \mathbf{coNP}$ entonces $R^c \in \mathbf{NP}$. Decidir instancia en L se limita entonces a preguntarle al oráculo si $(x, y) \in R^c$. Si la respuesta es negativa entonces M culmina con un estado de aceptación y si la respuesta es positiva entonces M culmina con un estado de negación. En consecuencia, $L = L\left(M^{R^c}\right)$.

Supongamos ahora que $L = L\left(M^B\right)$, donde M es una máquina de Turing no determinística con oráculo $B \in \mathbf{NP}$ que opera en tiempo polinomial y consideremos la máquina no determinística N que decide al lenguaje B en tiempo polinomial. Sea x una instancia de L. Bajo un vocabulario adecuado se puede construir una palabra y que codifica secuencias de computaciones de M asociadas a x, y computaciones de N asociadas a una palabra z escrita por M en la cinta de consulta. A R lo definimos como el conjunto de todos los pares (x, y) tales que y registra computaciones válidas de M^B en x, y computaciones válidas de N en z. Este conjunto es polinomialmente balanceado ya que M y N operan en tiempo polinomial y por lo tanto $|y|$ puede acotarse polinomialmente en función de $|x|$.

R puede decidirse en $\Pi_1^p = \mathbf{coNP}$. Para probar esto construiremos una máquina M' adecuada. Dada una instancia (x, y) de R, M' diagnostica sobre la palabra y si las computaciones de M^B en x son válidas, incluyendo aquellos pasos en los que aparezcan los estados de consulta $q_c, q_{sí}, q_{no}$ (todo esto puede verificarse determinísticamente en tiempo polinomial). Una vez se hayan verificado las computaciones de M, es decir, cuando se haya encontrado un estado de aceptación o negación para x, se procede a diagnosticar si las computaciones de N y el estado de respuesta $q_{sí}$ o q_{no} son válidos. Si y registra el estado $q_{sí}$, entonces también registrará alguna computación de N asociada a la palabra z y M' procede a verificar si estas computaciones son válidas (una vez más, todo esto puede hacerse determinísticamente en tiempo polinomial). Si y registra el estado q_{no}, entonces verificamos que z es una instancia negativa de B, pero esto es lo mismo que decidir si $z \in B^c$. Este es el único proceso no determinístico que haría la máquina M' y que por tanto clasificaría a $L(M') \in \mathbf{coNP}$. Resta ver que $R = L(M')$. Esto se deduce de lo siguiente. Todas las posibles maneras para que M' rechace a (x, y) son:

- Alguna computación de M o N, registrada en y, no es válida;
- y registra un estado de negación de x;
- y registra el estado q_{no}, pero $z \in B$;

- y registra el estado $q_{sí}$, pero la computación de N asociada a z culmina en un estado de negación.

Las últimas dos establecen que el estado de respuesta registrado en y debe ser acorde a la respuesta que daría el oráculo. En otras palabras, M' acepta (x, y) si y sólo si x es instancia positiva de L y y es una computación válida de M^B en x.

El caso $k > 1$ se demuestra usando inducción y un razonamiento similar al anterior. □

Definición 2.4. Sean L y R como en el teorema anterior y $x \in \Sigma^*$ una instancia de L. A una palabra $y \in \Sigma^*$ tal que $(x, y) \in R$ se le llama *certificado*, o *testigo*, de x. De esta manera, x es instancia positiva de L si y sólo si posee al menos un certificado y será instancia negativa si y sólo si no posee certificados.

Durante la construcción de la máquina M' en la demostración anterior, se realizó un proceso que vale la pena seguir analizando. Recordemos que dado un problema B en **NP**, el problema B^c puede ser reconocido en **coNP**. El teorema anterior sugiere que para decidir un lenguaje L en Σ_2^p, al menos un par de procesos no determinísticos deben realizarse, uno relacionado con **NP** y otro con **coNP**. En 1981, Chandra y Stockmeyer definieron un proceso computacional que generaliza el no determinismo usual y que además caracteriza a los niveles de la Jerarquía Polinomial bajo ciertos parámetros adecuados [7].

Definición 2.5. Una *máquina de Turing alternante* es una máquina cuyos estados están divididos en dos bloques: estados existenciales y estados universales. La noción de aceptación para este tipo de máquinas se define por inducción. Sea $M = (K, \Sigma, \Delta)$ una máquina alternante y x una entrada de M. Supongamos que para alguna computación $C = (q, a, p, b, d) \in \Delta$ asociada a x ocurre uno de los siguientes casos:

- p es un estado existencial y C da paso a una computación válida C' cuyo estado final es de aceptación;
- p es un estado universal, C da paso a al menos una computación válida C' y en todas éstas los estados finales son de aceptación.

Decimos entonces que M *acepta* a x. Sea $t(n)$ una función de complejidad propia. Denotamos por **ATIME**$[t(n)]$ a la clase de complejidad de todos los problemas L decidibles por máquinas de Turing alternantes que operan en tiempo $\mathcal{O}(t(n))$.

Note que las máquinas de **NP** son alternantes con estados existenciales únicamente. Así mismo, las máquinas de **coNP** son alternantes con estados universales únicamente. Estamos especialmente interesados en aquellas máquinas cuyos estados aparecen en determinado orden y realizan una cantidad acotada de alternaciones para cada entrada.

Definición 2.6. Sean $t(n)$ y $a(n)$ funciones de complejidad propia. La clase de complejidad **ATIME-ALT**$[t(n), a(n)]$ se define como la de todos los problemas L decidibles por máquinas alternantes que operan en tiempo $\mathcal{O}(t(n))$ y realizan a lo sumo $a(n)$ alternaciones entre estados existenciales y universales, comenzando por existenciales.

Como consecuencia inmediata de esta definición, tenemos la relación:

$$\mathbf{NP} = \bigcup_{m=1}^{\infty} \mathbf{ATIME\text{-}ALT}\left[n^m, 0\right] = \mathbf{ATIME\text{-}ALT}\left[n^{\mathcal{O}(1)}, 0\right].$$

Un resultado mucho más general es el siguiente:

Teorema 2.2. Para todo $k \geq 1$, $\Sigma_k^p = \mathbf{ATIME\text{-}ALT}\left[n^{\mathcal{O}(1)}, k-1\right]$.

Demostración. El caso $k = 1$ es justamente la observación anterior. Demostraremos el caso $k = 2$, el resto se deduce por un razonamiento inductivo. Sea $L \in \Sigma_2^p$. Debemos construir una máquina alternante M que decida L en tiempo polinomial y no realice más de una alternación entre estados existenciales y universales. Por Teorema 2.1, existe $R \in \mathbf{coNP}$ tal que

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

Sea M una máquina alternante que opera de la siguiente manera: dada una instancia x de L, M construye no determinísticamente una palabra y y valiéndose de ciertos estados clasificados como existenciales. Para construir esta palabra, M considera el tamaño de x y el hecho de que R es polinomialmente balanceada. Una vez construida esta palabra y, el estado de M cambia por un estado universal, el cual indica que se comenzará a determinar si $(x, y) \in R$. Para responder esto, M puede simular a la máquina que decide a R en **coNP**. De esta manera, M acepta a x si para alguna palabra y, todas las computaciones asociadas a (x, y) culminan en un estado de aceptación, pero esto último ocurre si y sólo si $x \in L$, de donde $L = L(M)$.

Supongamos ahora que $L \in \mathbf{ATIME\text{-}ALT}\left[n^{\mathcal{O}(1)}, 1\right]$ y M' es la máquina alternante que decide a L. Debemos construir una máquina no determinística $M^?$ que dotada de un

oráculo apropiado B decida el problema L. Sea x una instancia de L. Lo primero que hace M es generar no determinísticamente una palabra y que codifica computaciones de M' con estados existenciales y el último estado existencial da paso a una computación con estado universal. Dado que M' opera en tiempo polinomial, el tamaño de estas palabras dependen polinomialmente de $|x|$. Luego, M procede a verificar (determinísticamente y en tiempo polinomial) si y codifica computaciones válidas de M'. Una vez se haya verificado esto, M cambiará su estado por q_c. Sea C el conjunto de las palabras (x, y) tales que:

- y codifica correctamente una computación de M' en la entrada x;
- el último estado existencial en y da paso a un estado universal;
- todas las computaciones a partir de ese primer estado universal terminan en estados de aceptación.

C es decidible en **coNP**, lo cual se puede verificar definiendo una máquina que modele el proceso no determinístico universal de M'. Además, si definimos $B = C^c$ se obtiene que $L = L\left(M^B\right)$, pues una vez M haya consultado al oráculo si $(x, y) \in B$, lo único que tiene que hacer es aceptar x si la respuesta del oráculo es q_{no} y rechazar x si la respuesta del oráculo es $q_{sí}$. □

Aprovecharemos este último resultado para obtener una caracterización de Σ_2^p de mayor interés para este trabajo.

2.2. Caracterización Sintáctica de Σ_2^p

La demostración del Teorema 1.7 que aparece en [13] sugiere que la caracterización sintáctica de **NP** es mucho más restrictiva que como la planteó Fagin. Sea σ un vocabulario y $L = \text{MOD}[\Phi]$, con $\Phi \in \exists\mathbf{SO}[\sigma]$. Entonces $\Phi \equiv \exists S_1^{a_1} \cdots S_g^{a_g} \varphi$, donde φ es una sentencia de primer orden sobre el vocabulario $\sigma \cup \langle S_1^{a_1}, \ldots, S_g^{a_g} \rangle$ con todas las variables cuantificadas universalmente, es decir, $\varphi \equiv \forall x_1 \cdots x_k \varphi_1(x_1, \ldots, x_k)$, donde φ_1 es una fórmula de primer orden sin cuantificadores. Además, φ_1 se puede considerar en FNC y en cada cláusula no posee más de un literal propio del vocabulario σ.[2] En el caso del Corolario 1.8, las fórmulas de segundo orden se restringen por dualidad.

[2] Esto último lo explica Immerman como consecuencia de reducir a dos la cantidad de decisiones tomadas por una máquina de Turing no determinística, teniendo que decidir entre efectuar o no cierta computación.

Se pueden aprovechar algunas de estas propiedades para caracterizar sintácticamente cada uno de los niveles de la Jerarquía Polinomial. Nos limitamos a mostrar el resultado sólo para el segundo nivel y los demás se obtienen por un razonamiento inductivo. Denotamos por $\exists\forall\mathbf{SO}$ al extracto de **SO** que contiene a todas las sentencias de segundo orden cuyas variables relacionales se encuentran cuantificadas en dos bloques, el primer bloque cuantificado existencialmente y el segundo universalmente.

TEOREMA 2.3. $\Sigma_2^p = \exists\forall\mathbf{SO}$.

DEMOSTRACIÓN. Sea σ un vocabulario y sea la fórmula de segundo orden

$$\Phi \equiv \exists S_1^{a_1} \cdots S_g^{a_g} \forall T_1^{b_1} \cdots T_h^{b_h} \varphi(\overline{x}), \tag{2.1}$$

donde $S_1, \ldots, S_g, T_1, \ldots, T_h$ son variables relacionales y $\varphi(\overline{x})$ es una sentencia de primer orden sobre el vocabulario $\sigma \cup \langle S_1, \ldots, S_g, T_1, \ldots, T_h \rangle$. Sea $\mathcal{A} \in \text{STRUC}[\sigma]$. Construiremos una máquina alternante que decida si $\mathcal{A} \models \Phi$. Recordemos que esto último sucede si y sólo si existen relaciones $S_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ con $i = 1, \ldots, g$ tales que para todas las relaciones $T_j^{\mathcal{A}} \subseteq |\mathcal{A}|^{b_j}$ con $j = 1, \ldots, h$ se cumple que

$$\langle \mathcal{A}, S_1^{\mathcal{A}}, \ldots, S_g^{\mathcal{A}}, T_1^{\mathcal{A}}, \ldots, T_h^{\mathcal{A}} \rangle \models \varphi(\overline{x}).$$

Sea M la máquina de Turing que opera de la siguiente manera: cuando se suministre la entrada $\mathcal{A}$ (codificada convenientemente, en binario por ejemplo, como en la PROPOSICIÓN 1.5), M comienza a generar no determinísticamente palabras que codifican a las relaciones S_i. Como el tamaño de estas relaciones depende polinomialmente de $\|\mathcal{A}\|$, estas codificaciones también son de tamaño polinomial. Todo este proceso se realiza por medio de unos estados clasificados como existenciales. Una vez se hayan construido todas estas relaciones, el estado de M cambia por un estado universal cuya función es anunciar un segundo proceso no determinístico universal, esta vez para generar codificaciones de las relaciones T_j. Construidas estas relaciones, M empieza un proceso determinístico. Recordemos que una fórmula de primer orden es decidible determinísticamente en espacio logarítmico y por tanto, en tiempo polinomial. Por la definición de máquinas alternantes, M acepta a la entrada $\mathcal{A}$ si y sólo si existen codificaciones de S_i tales que para todas las codificaciones de T_j, la verificación de la sentencia $\varphi(\overline{x})$ culmina en un estado de aceptación, es decir, M acepta a $\mathcal{A}$ si y sólo si $\mathcal{A} \models \Phi$. En otras palabras, $\text{MOD}[\Phi] = \{\mathcal{A} \in \text{STRUC}[\sigma] : \mathcal{A} \models \Phi\} = L(M)$.

Ahora, sea $L \subseteq \text{STRUC}[\sigma]$ un problema en Σ_2^p. Veamos que existe una fórmula de segundo orden $\Phi \in \exists\forall\mathbf{SO}$ tal que $L = \text{MOD}[\Phi]$. Sea M la máquina alternante que decide a L usando los recursos de $\mathbf{ATIME\text{-}ALT}[n^{\mathcal{O}(1)}, 1]$. Podemos pensar las computaciones existenciales de M de la siguiente manera: dada una instancia $\mathcal{A}$ de L, M reconoce como entrada a $x = \text{bin}(\mathcal{A})$ y comienza a construir no deterministicamente una palabra binaria y tal que $|y| \leq |x|^d$ para algún entero positivo d. Este valor d se puede fijar, pues el Teorema 2.1 lo garantiza. Los siguientes pasos de M son universales y sirven para determinar si las computaciones asociadas a y culminan en estados de aceptación. Note que esto último es un proceso en **coNP**. Es decir, al escribir la palabra y, M puede verse como una máquina de **coNP** computando sobre la entrada xy. Por el Corolario 1.8, existe una sentencia $\Psi \in \forall\mathbf{SO}$ tal que M acepta a xy si y sólo si la estructura asociada a xy satisface Ψ. ¿Qué estructura representa xy exactamente?

Consideremos un símbolo relacional S^d que no sea parte de σ y definamos $S^{\mathcal{A}} \subseteq |\mathcal{A}|^d$ como el conjunto cuya codificación binaria concuerda con la palabra y, al menos en los primeros $|y|$ bits. Decimos entonces que $\Psi \in \forall\mathbf{SO}[\sigma \cup \langle S^d \rangle]$. Como S es construido por M bajo condiciones existenciales, entonces M acepta a $\mathcal{A}$ si y sólo si $\mathcal{A} \models \exists S^d \Psi$. □

La fórmula $\varphi(\overline{x})$ en la sentencia (2.1) se puede asumir con ciertas características especiales dadas por la observación hecha al comienzo de esta sección. Es decir, podemos asumir que $\varphi(\overline{x})$ es una sentencia existencial de primer orden en FND y en cada implicante no aparece más de un literal propio de σ. Siguiendo con las mismas ideas desarrolladas en la última demostración, se puede hallar una caracterización de Σ_3^p, Σ_4^p y así sucesivamente. En consecuencia,

Corolario 2.4. Un problema de decisión se encuentra en **PH** si y sólo si es expresable en segundo orden, es decir, $\mathbf{PH} = \text{SO}$.

También podemos caracterizar por dualidad a las clases Π_k^p. Mostramos sólo el resultado para $k = 2$, el resto se deduce por analogía.

Corolario 2.5. $\Pi_2^k = \forall\exists\mathbf{SO}$.

2.3. Problemas en Σ_2^p

Aprovecharemos esta sección para introducir varios problemas pertenecientes a la clase de complejidad Σ_2^p. La mayoría de estos problemas aparecen listados en [**23**], junto con las referencias de los trabajos originales donde se conjeturó o se demostró su completitud. Nuestro propósito es demostrar que también son completos vía proyecciones, pero esta tarea la dejamos para los próximos capítulos. En los textos dedicados a la teoría de complejidad computacional, es usual utilizar pseudocódigos para describir el algoritmo que clasifica a los problemas de decisión. Nosotros usaremos como principal herramienta el Teorema 2.3.

Muchos de los problemas pertenecientes a las clases de complejidad Σ_2^p o Π_2^p se pueden ver como generalizaciones o versiones de problemas en **NP**. La versión del problema de satisfabilidad en Σ_2^p estudiada por Stockmeyer es

$\exists\forall$SAT

Instancia: Fórmula Booleana ϕ en FND, con variables clasificadas distintivamente como existenciales y universales.

Pregunta: ¿Existe una asignación de valores de las variables existenciales tal que ϕ se satisface para toda asignación de valores de las variables universales?

Proposición 2.6. $\exists\forall$SAT se encuentra en Σ_2^p.

Demostración. Demostraremos que $\exists\forall$SAT es expresable en $\exists\forall$**SO**. Consideremos el vocabulario $\sigma'_{\text{fnd}} = \langle E^1, Q^2, M^2\rangle$ y sea ϕ una instancia de $\exists\forall$SAT. A ϕ la podemos asociar con una σ'_{fnd}-estructura $\mathcal{A}$ como sigue:

- $|\mathcal{A}| = [n]$, donde n es el máximo entre el número de variables Booleanas e implicantes de ϕ. Por conveniencia, imponemos que el número n siempre coincida con la segunda condición.
- $E^{\mathcal{A}} \subseteq |\mathcal{A}|$ indica cuáles variables de ϕ se consideran existenciales.
- Los conjuntos $Q^{\mathcal{A}}$ y $M^{\mathcal{A}}$, ambos contenidos en $|\mathcal{A}|^2$, se interpretan exactamente igual como en el Ejemplo 1.4.

Por ejemplo, la σ'_{fnd}-estructura $\mathcal{A}_1 = \{[4], E, Q, M\}$, con

$$E = \{0, 1\}, \qquad Q = \{(0,0), (1,0), (1,1), (1,2), (3,1)\}, \qquad M = \{(0,1), (2,0), (2,2), (3,2)\},$$

codifica a la fórmula Booleana

$$\phi_1 \equiv (x_1 \wedge \neg x_2) \vee (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_3) \vee (x_2 \wedge \neg x_3),$$

cuyas variables existenciales son x_1 y x_2, interpretadas como los elementos 0 y 1 de $|\mathcal{A}|$.

Recordemos que una asignación de valores de verdad la podemos ver como una relación de aridad 1 en la cual se especifican las variables cuyo valor asignado es **true**. En este caso, necesitamos dos de tales relaciones, una para las variables existenciales y otra para las variables universales. Dado que las fórmulas Booleanas se encuentran en forma normal disyuntiva, una instancia ϕ se satisface si y sólo si existe una asignación de valores de las variables existenciales tal que para todas las asignaciones de las variables universales, hay al menos un implicante con valor **true**, es decir, todos los literales en ese implicante tienen valor asignado **true**. Todo esto lo expresamos en segundo orden de la siguiente manera:

$$\Psi_{\text{sat}} \equiv \exists S^1 \forall T^1 \left[\psi_1 \wedge \psi_2 \wedge \psi_3\right], \tag{2.2}$$

donde

$$\begin{aligned}
\psi_1 &\equiv \forall x\big[(S(x) \rightarrow E(x)) \wedge (T(x) \rightarrow \neg E(x))\big],\\
\psi_2 &\equiv \exists x \forall y\big[(Q(x,y) \wedge E(y) \rightarrow S(y)) \wedge (Q(x,y) \wedge \neg E(y) \rightarrow T(y))\big],\\
\psi_3 &\equiv \exists x \forall y\big[(M(x,y) \wedge E(y) \rightarrow \neg S(y)) \wedge (M(x,y) \wedge \neg E(y) \rightarrow \neg T(y))\big].
\end{aligned}$$

En ψ_1 se especifica que la relación S es subconjunto de E, específicamente, el conjunto de variables existenciales con valor asignado **true**. T por su parte representa la asignación **true** de las variables universales. Con ψ_2 y ψ_3 se verifica que hay al menos un implicante certificado. Luego, si ϕ es instancia de $\exists\forall$SAT y $\mathcal{A}$ es la σ'_{fnd}-estructura que la codifica, entonces

$$\phi \in \exists\forall\text{SAT} \qquad \Leftrightarrow \qquad \mathcal{A} \models \Psi_{\text{sat}},$$

quedando demostrado que $\exists\forall\text{SAT} = \text{MOD}[\Psi_{\text{sat}}]$. □

Por dualidad, el problema $\forall\exists$SAT, con las instancias restrictas a fórmulas en FNC, se encuentra en Π_2^p. Presentaremos ahora otras versiones de los problemas de la lógica Booleana. No seremos tan rigurosos al caracterizarlos mediante fórmulas de segundo orden, en algunos casos omitiremos incluso la expresión explícita de ciertas fórmulas.

Ejemplo 2.1. El problema de insatisfabilidad para fórmulas Booleanas en FNC adquiere un nivel superior de complejidad cuando consideramos variables existenciales. Considérese el siguiente problema:

∃UNSAT

Instancia: Fórmula Booleana ϕ en FNC, con variables clasificadas distintivamente como existenciales y universales.

Pregunta: ¿Existe una asignación de valores de las variables existenciales tal que ϕ no se satisface para ninguna asignación de valores de las variables universales?

El vocabulario a considerar es, naturalmente, $\sigma'_{\text{fnc}} = \langle E^1, P^2, N^2 \rangle$. La fórmula Ψ_{unsat} que caracteriza a ∃UNSAT es muy parecida a la sentencia (2.2), salvo que se sustituyen los símbolos relacionales Q y M por N y P, respectivamente. □

Ejemplo 2.2. Introducimos ahora el símbolo de cuantificación ∃!, que indica existencia y unicidad.

∃∃!SAT[3]

Instancia: Fórmula Booleana ϕ en FNC, con algunas de sus variables clasificadas como existenciales.

Pregunta: ¿Existe una asignación de valores de las variables existenciales que se pueda extender a un único certificado de ϕ?

Podemos volver a utilizar el vocabulario σ'_{fnc} del ejemplo anterior para codificar instancias de ∃∃!SAT. Consideremos la siguiente fórmula:

$$\Psi_{\text{unique}} \equiv \exists S^1 T^1 \forall U^1 \big[\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4\big], \tag{2.3}$$

donde ϕ_1 verifica que S es una asignación de valores de las variables existenciales, mientras que T y U son asignaciones de las variables no existenciales. ϕ_2 es equivalente a la sentencia

$$\exists y \big[T(y) \leftrightarrow \neg U(y)\big],$$

[3]No se sabe si ∃!SAT (el problema de verificar si una fórmula Booleana tiene un único certificado) es Σ_2^p-completo, se conjetura que no [22], por eso requerimos un nivel de cuantificación extra.

que implica que T y U son asignaciones distintas. La sentencia ϕ_3 depende de las variables relacionales S y T, y verifica que ambas forman un certificado, mientras que ϕ_4 depende de S y U, y verifica que ambas no forman un certificado. Las propiedades ϕ_1, ϕ_3 y ϕ_4 son adaptaciones de las subfórmulas ψ_1, ψ_2 y ψ_3 de (2.2). □

Los problemas relacionados con satisfabilidad Booleana no son los únicos que nos interesan. Introducimos ahora varios problemas relacionados con teoría de grafos y que serán indispensables más adelante.

Ejemplo 2.3. Cuando se consideran dos o más procesos computacionales no determinísticos y alternantes, muchos problemas de **NP** se pueden adaptar a juegos entre dos contrincantes. Un ejemplo de ello es el problema de *colorabilidad*.

Sea un grafo $\mathcal{G} = \langle V, A\rangle$ y un subconjunto $U \subset V$. Una k-*coloración* de U es una partición en k bloques (no vacíos) de manera que no haya vértices adyacentes en un mismo bloque. Una k-coloración del grafo $\mathcal{G}$ es una k-coloración de V. Es bien sabido que reconocer si un grafo es 2-colorable puede hacerse determinísticamente en tiempo polinomial, mientras que 3-colorabilidad define un problema **NP**-completo. Con el conjunto U se alcanza un nivel extra de complejidad. Nos restringimos al caso $k = 3$ y al hablar de *colorabilidad* nos referimos a 3-colorabilidad. Consideremos el siguiente problema:

CERTAIN VICTORY
Instancia: Grafo $\mathcal{G} = \langle V, A\rangle$ y un subconjunto $U \subset V$.
Pregunta: ¿Existe una coloración de U que no pueda extenderse a una coloración de $\mathcal{G}$?

El nombre del problema surge de lo siguiente: Sean $\mathcal{G} = \langle V, A\rangle$, $U \subset V$ y un par de jugadores que llamaremos simplemente Jugador 1 y Jugador 2. Al Jugador 1 se le otorga el conjunto U, mientras que al Jugador 2 se le otorga el complemento. El desarrollo del juego es el siguiente: Jugador 1 propone una coloración válida de U (no hay vértices adyacentes con el mismo color) y luego Jugador 2 trata de extender ésta a una coloración de $\mathcal{G}$. Si Jugador 2 lo logra, gana, de lo contrario, gana Jugador 1. La pregunta "¿habrá una coloración válida propuesta por Jugador 1 de modo que Jugador 2 no pueda ganar?" es otra manera de plantear el problema CERTAIN VICTORY.

El vocabulario a usar para describir este problema es $\sigma'_g = \langle U^1, A^2\rangle$. Consideremos la fórmula

$$\Psi_{\text{victory}} \equiv \exists F^1 G^1 H^1 \forall I^1 J^1 K^1 \big[\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5\big], \tag{2.4}$$

donde

$$\phi_1 \equiv \forall x\big[(F(x) \to U(x)) \wedge (G(x) \to U(x)) \wedge (H(x) \to U(x))\big],$$
$$\phi_2 \equiv \forall x\big[U(x) \to F(x) \oplus G(x) \oplus H(x)\big],$$
$$\phi_3 \equiv \forall xy\big[U(x) \wedge U(y) \wedge A(x,y) \to \neg\big((F(x) \wedge F(y)) \vee (G(x) \wedge G(y)) \vee (H(x) \wedge H(y))\big)\big],$$
$$\phi_4 \equiv \forall x\big[(I(x) \oplus J(x) \oplus K(x)) \wedge (F(x) \to I(x)) \wedge (G(x) \to J(x)) \wedge (H(x) \to K(x))\big],$$
$$\phi_5 \equiv \exists xy\big[A(x,y) \wedge \big((I(x) \wedge I(y)) \vee (J(x) \wedge J(y)) \vee (K(x) \wedge K(y))\big)\big].$$

El símbolo $\oplus$ representa el "o" excluyente. Las fórmulas ϕ_1 y ϕ_2 dicen que F, G y H forman una partición de U, y la fórmula ϕ_3 indica que se trata de una coloración válida. La fórmula ϕ_4 dice que I, J y K son las respectivas extensiones de los colores F, G y H sobre todo el grafo y la fórmula ϕ_5 indica que esta extensión no es una coloración del grafo. □

EJEMPLO 2.4. Sea $\mathcal{G} = \langle V, A\rangle$ un grafo. Un *clique* de $\mathcal{G}$ es un subgrafo completo y decimos que es *maximal* si no existe otro clique que lo contenga. El siguiente problema será sumamente relevante en los próximos capítulos:

2 CLIQUE COLORING, a.k.a, 2CC
Instancia: Grafo $\mathcal{G} = \langle V, A\rangle$.
Pregunta: ¿Existe una 2-coloración sobre V tal que ningún clique maximal de $\mathcal{G}$ sea monocromático?

En la FIGURA 2.1 se muestran un par de instancias de 2CC, la primera positiva y la segunda negativa. El segundo grafo (el ciclo de cinco vértices) es el "menor" grafo (conexo o disconexo) que no satisface la propiedad 2CC. Con "menor" queremos decir que es el grafo con la menor cantidad de vértices y aristas que satisface determinada propiedad. Más adelante nos expresaremos bajo los mismos términos.

El lenguaje a utilizar es sencillamente σ_g. Antes de escribir la fórmula que caracteriza a 2CC, expresaremos ciertas propiedades de grafos. Sea $\mathcal{H} = \langle W, B\rangle$ un subgrafo de $\mathcal{G}$.

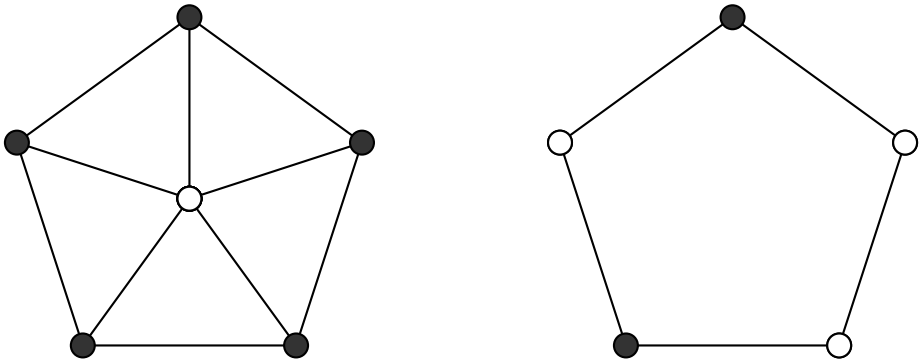

FIGURA 2.1. *Instancias de* 2CC.

Considerando a W y B como símbolos relacionales, $\mathcal{H}$ satisface la fórmula

$$\text{subgraph}(W, B) \equiv \forall xy\big[B(x, y) \to W(x) \wedge W(y) \wedge A(x, y)\big].$$

Podemos definir una fórmula complete(W, B), como en el EJEMPLO 1.3, que decide si $\mathcal{H}$ es completo y será maximal si además satisface

$$\text{máx}(W) \equiv \forall x\exists z\big[\neg W(x) \to W(z) \wedge \neg A(x, z)\big].$$

Llamemos maxclique(W, B) a la conjunción de las tres fórmulas descritas. Supongamos que tenemos una 2-coloración de V, y escojamos los colores rojo y azul como las etiquetas de tal partición. Basta con describir esta coloración como un subconjunto R de V de aquellos vértices que se colorean de rojo. Decimos que el subgrafo $\mathcal{H}$ es *monocromático rojo* si todos los vértices de W están en R, y es *monocromático azul* si ninguno está en R. Considerando R como un símbolo relacional, $\mathcal{H}$ es monocromático si satisface la fórmula

$$\text{monochrome}(W, R) \equiv \forall xy\big[W(x) \wedge W(y) \to (R(x) \leftrightarrow R(y))\big],$$

la cual expresa que todos los vértices en W son rojos o azules, a la vez. Finalmente, el grafo $\mathcal{G}$ cumple la propiedad 2CC si y sólo si satisface la fórmula de segundo orden

$$\Psi_{2cc} \equiv \exists R^1 \forall W^1 B^2\big[\text{maxclique}(W, B) \to \neg\text{monochrome}(W, R)\big]. \tag{2.5}$$

□

Por último, describiremos un problema con valor y costo. En [4] se definen este tipo de problemas formalmente.

Ejemplo 2.5. Usaremos la propiedad ∃∀SAT como base, pero podríamos utilizar algún otro como ∃UNSAT, ∃∃!SAT o CERTAIN VICTORY. Sea $\mathcal{A} = \langle [n], Q, M\rangle$ una σ_{fnd}-estructura. Supongamos que cada elemento x de $[n]$ tiene asociado un *valor* $\nu(x)$, codificado en binario. Sea K un número, codificado en binario, que llamaremos *costo*. Ninguno de estos términos tiene que ser necesariamente un elemento de $[n]$, por lo que cada uno de ellos nos conviene pensarlos como en el Ejemplo 1.2. Consideremos el siguiente planteamiento:

> VC SAT
> **Instancia:** $\mathcal{A} = \langle [n], Q, M\rangle$ una σ_{fnd}-estructura, un valor $\nu(x)$ para cada variable Booleana x, y un costo K.
> **Pregunta:** ¿Existe un subconjunto E de variables tal que la suma de todos los valores de elementos en E no sea mayor que K y de modo que $\mathcal{A} \cup \langle E\rangle$ sea una instancia positiva de ∃∀SAT?

El problema consiste en saber si bajo cierto costo podemos definir un conjunto E cuyos elementos se comporten como variables Booleanas existenciales. El vocabulario a considerar es $\sigma_{\text{fnd}} \cup \langle V^2, K^1\rangle$. Una estructura $\mathcal{A}$ sobre este vocabulario es una fórmula Booleana en FND con $K^{\mathcal{A}}$ como la codificación binaria del costo y $V^{\mathcal{A}}$ una relación binaria tal que $(i, j) \in V$ si y sólo si hay un 1 en el j-ésimo bit del valor asociado a la variable Booleana i, es decir, con la variable j se codifica en binario a $\nu(i)$. Superficialmente, la fórmula de segundo orden que describe a este problema tiene la forma

$$\Psi_{\text{vc}} \equiv \exists E^1 \left[\sum_{x \in E} \nu(x) \leq K \wedge \Psi_{\text{sat}}(E, Q, M) \right], \tag{2.6}$$

donde Ψ_{sat} es la fórmula (2.2). Para describir la desigualdad mediante una fórmula de primer orden, usaremos un símbolo relacional auxiliar S^2, que codificará en binario la suma de los valores según el orden natural restricto a E. El símbolo de desigualdad que aparece en la fórmula no es el símbolo numérico de orden, sino una comparación bit a bit de un par de representaciones binarias.

La siguiente fórmula verifica cuál es el mínimo elemento de E:

$$\text{mín}(x, E) \equiv E(x) \wedge \forall y \big(y < x \to \neg E(y)\big).$$

Análogamente, podemos definir el elemento máximo de E. La propiedad de sucesor en E se verifica como sigue:

$$\mathrm{suc}(x, y, E) \equiv E(x) \wedge E(y) \wedge \forall z\big(x < z < y \rightarrow \neg E(z)\big).$$

La propiedad de suma se describirá por partes. Básicamente, se requieren un par de fórmulas, la primera para establecer un valor inicial para la suma S, y la segunda para sumar sucesivamente los valores de E.

$$\varphi_1(x, E, S) \equiv \mathrm{mín}(x, E) \rightarrow \forall j\big(S(x, j) \leftrightarrow V(x, j)\big),$$

$$\varphi_2(x, y, E, S) \equiv \mathrm{suc}(x, y, E) \rightarrow \forall j\big(S(y, j) \leftrightarrow S(x, j) \oplus V(y, j) \oplus \varphi_{\mathrm{carry}}(j)\big),$$

donde $\oplus$ es el "o" excluyente y

$$\varphi_{\mathrm{carry}}(j) \equiv \exists k\Big(k < j \wedge S(x, k) \wedge V(y, k) \wedge \forall i\big(k < i < j \rightarrow S(x, i) \oplus V(y, i)\big)\Big).$$

Por último, debemos comparar bit a bit el costo con la suma total. Ello se verifica con la siguiente fórmula:

$$\varphi_3(x, E, S) \equiv \mathrm{máx}(x, E) \rightarrow \exists j\Big(\neg S(x, j) \wedge K(j) \wedge \forall k\big(j < k \rightarrow S(x, k) \wedge K(k)\big)\Big).$$

De esta manera,

$$\Psi_{\mathrm{vc}} \equiv \exists E^1 S^2\Big[\forall x y\big(\varphi_1(x, E, S) \wedge \varphi_2(x, y, E, S) \wedge \varphi_3(x, E, S)\big) \wedge \Psi_{\mathrm{sat}}(E, Q, M)\Big]. \tag{2.7}$$

□

Capítulo 3

Completitud en Σ_2^p

Stockmeyer probó que $\exists\forall$SAT es completo en Σ_2^p vía reducciones logarítmicas. A continuación demostraremos que este problema es completo vía proyecciones. Esta demostración se basa en un análisis similar al utilizado por Immerman para demostrar la completitud de SAT en **NP**.

Teorema 3.1. $\exists\forall$SAT es Σ_2^p-completo vía proyecciones.

Demostración. Sea σ un vocabulario y $A \subseteq \text{STRUC}[\sigma]$ un problema en Σ_2^p. Como ya sabemos que $\exists\forall$SAT está en Σ_2^p sólo nos resta ver que $A \leq_{\text{fop}} \exists\forall$SAT. Por Teorema 2.3 existe una fórmula Φ en $\exists\forall\mathbf{SO}[\sigma]$ tal que $A = \text{MOD}[\Phi]$. Consideremos a Φ como en la fórmula (2.1):

$$\Phi \equiv \exists S_1^{a_1} \cdots S_g^{a_g} \forall T_1^{b_1} \cdots T_h^{b_h} \exists x_1 \cdots x_c \varphi, \tag{2.1}$$

y sin pérdida de generalidad, supongamos que

$$\varphi(x_1, \ldots, x_c) \equiv \bigvee_{i=1}^{r} I_i(x_1, \ldots, x_c),$$

donde cada I_i es equivalente a un implicante $(L_1 \wedge \ldots \wedge L_{\ell_i})$. Los literales L son fórmulas atómicas dadas por relaciones numéricas, relaciones propias de σ, relaciones existenciales $S_1, \ldots, S_g$, o relaciones universales $T_1, \ldots, T_h$. En estos dos últimos casos diremos que L es un literal existencial o universal, según el caso. Esencialmente, estamos pidiendo que $\varphi(x_1, \ldots, x_c)$ esté representada en FND. Se puede asumir también que en cada implicante I_i no aparece más de un literal de σ (ver la nota al pie de página de Pag. 40).

Ahora, dada una instancia $\mathcal{A}$ del problema A, debemos interpretarla apropiadamente como una fórmula booleana $I(\mathcal{A})$ de manera que se satisfaga la propiedad

$$\mathcal{A} \in A \qquad \Leftrightarrow \qquad I(\mathcal{A}) \in \exists\forall\text{SAT} \tag{3.1}$$

y aprovecharemos la fórmula $\exists x_1 \cdots x_c \varphi$ para hacer esto. Describiremos a $I(\mathcal{A})$ por medio del vocabulario σ'_{fnd}. Primero identificaremos el dominio de $I(\mathcal{A})$ y sus elementos:

- $|I(\mathcal{A})| = |\mathcal{A}|^k$, donde $k = \log(m) + c$, siendo $m = \max\{g + h, r\}$;
- cada tupla $\overline{x} \in |I(\mathcal{A})|$ puede verse como la concatenación de un par de tuplas $\overline{x}_1$ y $\overline{x}_2$ de tamaños $\log(m)$ y c, respectivamente;
- a los implicantes $I_i(\overline{x}_2)$ los interpretamos con una tupla $\overline{x}_1\overline{x}_2$ donde $\overline{x}_1$ codifica en binario al índice i;
- a los literales $S_i(\overline{x}_2)$ y $T_j(\overline{x}_2)$ los interpretamos como variables Booleanas: si quisiéramos referirnos a $S_i(\overline{x}_2)$ tomamos la tupla $\overline{x}_1\overline{x}_2$ con $\overline{x}_1$ codificando al índice i en binario y si quisiéramos referirnos a $T_j(\overline{x}_2)$ tomamos la tupla $\overline{x}_1\overline{x}_2$ con $\overline{x}_1$ codificando a $g + j$ en binario.

La notación $L(\overline{x}_2)$ quiere decir que estamos evaluando al literal L en las entradas de la c-tupla $\overline{x}_2$ como lo indica φ. Ahora, una instancia $\mathcal{A}$ de A es positiva si y sólo si para algún índice $i = 1, \ldots, r$ y alguna c-tupla $\overline{x}_2$, $\mathcal{A} \models I_i(\overline{x}_2)$, o en otras palabras, $\mathcal{A}$ satisface todos los literales $L(\overline{x}_2)$ en el implicante $I_i(\overline{x}_2)$. Los implicantes de $I(\mathcal{A})$ estarán determinadas por aquellos $I_i(\overline{x}_2)$ tales que su satisfabilidad dependa necesariamente de los literales existenciales y universales. En función de estas ideas construiremos las relaciones $E^{I(\mathcal{A})}$, $Q^{I(\mathcal{A})}$ y $M^{I(\mathcal{A})}$.

El conjunto $E^{I(\mathcal{A})}$ es fácil de describir, pues es expresable mediante una simple fórmula de primer orden numérica. La tupla $\overline{x}_1\overline{x}_2$ está en E si y sólo si $\overline{x}_1$ codifica en binario a algún número $i \leq g$. En resumen, E está determinado por la fórmula

$$\varphi_E(\overline{x}_1\overline{x}_2) \equiv (\exists i)(i \leq g \wedge \overline{x}_1 = \text{bin}(i)).$$

Aquí $\text{bin}(i)$ denota la codificación binaria de i en $\log(m)$-bits y $\big(\overline{x}_1 = \text{bin}(i)\big)$ es la igualdad bit a bit. El conjunto $Q^{I(\mathcal{A})}$ requiere un poco más de análisis. Supongamos que el implicante $I_i(\overline{x}_2)$ contiene la subfórmula $\big(\alpha \wedge R(\overline{x}_2) \wedge L(\overline{x}_2)\big)$, donde α es la conjunción de todos los literales numéricos que aparecen en I_i, $R(\overline{x}_2)$ es el único literal de σ que aparece en I_i y L es cualquier literal existencial o universal positivo en I_i. Si la instancia $\mathcal{A}$ es tal que $\mathcal{A} \not\models \alpha \wedge R(\overline{x}_2)$, no haría falta referirse al literal $L(\overline{x}_2)$ pues sin importar si éste se satisface o no, $\mathcal{A} \not\models I_i(\overline{x}_2)$, por lo que los implicantes donde esto suceda serán descartados. De esta manera, el par $(\overline{x}_1\overline{x}_2, \overline{y}_1\overline{y}_2)$ está en Q si y sólo si $\overline{x}_1\overline{x}_2$ codifica a un implicante $I_i(\overline{x}_2)$ tal que su literal positivo $L(\overline{x}_2)$ podría ser relevante para su satisfabilidad. En resumen, una parte de Q está determinada por la disyunción de las fórmulas

$$(\overline{x}_1 = \text{bin}(i)) \wedge (\overline{y}_1 = \text{bin}([\ell])) \wedge (\overline{x}_2 = \overline{y}_2) \wedge \alpha \wedge R(\overline{y}_2),$$

donde el valor ℓ es el índice que le corresponde al literal L como variable relacional y $[\ell]$ indica que es ℓ si L es existencial o $g+\ell$ si L es universal. Otra parte de Q la determinan las cláusulas en las que no hayan relaciones propias de σ, es decir, fórmulas muy parecidas a la anterior pero sin $R(\overline{y}_2)$. Denotemos por φ_Q a la disyunción de todas las fórmulas descritas.

De manera similar, podemos construir una fórmula φ_M que caracterice a la relación $M^{I(\mathcal{A})}$, salvo que ahora L representa un literal existencial o universal negativo en determinado implicante.

Es de notar que todas las fórmulas mencionadas hasta ahora son proyecciones (1.6). Además, la reducción $I = \lambda_{\overline{x},\overline{y}}\langle \mathbf{true}, \varphi_E, \varphi_Q, \varphi_M\rangle$ fue construida de manera que se satisfaga la relación (3.1). □

Como la relación $\leq_{fop}$ es transitiva, para ver que un problema B en Σ_2^p es completo, basta demostrar que $\exists\forall\text{SAT} \leq_{fop} B$.

Ejemplo 3.1. Las propiedades de fórmulas Booleanas vistas en el primer capítulo permiten construir una reducción natural de $\exists\forall$SAT a $\exists$UNSAT. El análisis es similar al del Ejemplo 1.6, por lo que sólo nos limitamos a mostrar la reducción. Sea $\mathcal{A}$ una σ'_{fnd}-estructura y definamos $I(\mathcal{A})$ como la siguiente σ'_{fnc}-estructura:

- $|I(\mathcal{A})| = |\mathcal{A}|$;
- $E^{I(\mathcal{A})} = E^{\mathcal{A}}$, el cual está asociado a la fórmula proyectiva $\varphi_E(x) \equiv E(x)$;
- $P^{I(\mathcal{A})} = M^{\mathcal{A}}$, el cual está asociado a la fórmula proyectiva $\varphi_P(x,y) \equiv M(x,y)$;
- $N^{I(\mathcal{A})} = Q^{\mathcal{A}}$, el cual está asociado a la fórmula proyectiva $\varphi_N(x,y) \equiv Q(x,y)$.

De esta manera, $I = \lambda_{xy}\langle \mathbf{true}, \varphi_E, \varphi_P, \varphi_N\rangle$ es una proyección de $\exists\forall$SAT a $\exists$UNSAT. □

Ejemplo 3.2. Mostramos ahora una reducción de $\exists$UNSAT a $\exists\exists!$SAT. La propiedad que nos permitirá clasificar a éste último como Σ_2^p-completo es la siguiente: la fórmula Booleana $\phi(y_1,\ldots,y_m)$ es insatisfactible si y sólo si

$$(z \vee \phi(y_1,\ldots,y_m)) \wedge (\neg z \vee y_1) \wedge \cdots \wedge (\neg z \vee y_m) \tag{3.2}$$

tiene una única solución (específicamente, aquella que asigna el valor **true** a todas las variables y_i y a la nueva variable z). Note que si $\phi(y_1,\ldots,y_m)$ está en FNC entonces podemos asumir que (3.2) también, ya que se puede distribuir la nueva variable z en cada una de las cláusulas de ϕ.

Sea $\mathcal{A}$ una σ'_{fnc}-estructura, vista ésta como instancia de $\exists$UNSAT. Debemos construir una σ'_{fnc}-estructura $I(\mathcal{A})$, de modo que $\mathcal{A} \in \exists$UNSAT si y sólo si $I(\mathcal{A}) \in \exists\exists!$SAT. $I(\mathcal{A})$ no será más que la codificación de (3.2). Supongamos que el dominio de $\mathcal{A}$ es $[n]$ para algún natural n y definamos

- $|I(\mathcal{A})| = \{(i,j) \in \mathcal{A} : i = 0 \text{ o } i = 1\}$, es decir, el dominio de $I(\mathcal{A})$ posee $2n$ elementos;
- los pares $(0,x)$ serán las interpretaciones de las variables x de $\mathcal{A}$;
- a la variable z la asociamos al par $(1,0)$;
- las primeras n cláusulas de $I(\mathcal{A})$ serán iguales a las de $\mathcal{A}$ salvo que en cada una de ellas se debe incluir a la variable z;
- $(\neg z \vee y)$ será cláusula de $I(\mathcal{A})$ si y sólo si y es una variable universal de $\mathcal{A}$.

Teniendo en cuenta todas estas condiciones describiremos explícitamente a los conjuntos $E^{I(\mathcal{A})}$, $P^{I(\mathcal{A})}$ y $N^{I(\mathcal{A})}$. La fórmula numérica $\psi_0(x,y) \equiv (x = 0 \vee x = 1)$ describe el dominio.

Al conjunto $E^{I(\mathcal{A})}$ lo asociamos con la siguiente fórmula:

$$\psi_E(x,y) \equiv \big[x = 1 \wedge y \neq 0\big] \vee \big[x = 0 \wedge E(y)\big].$$

El conjunto $P^{I(\mathcal{A})}$ se asocia con la siguiente fórmula:

$$\begin{aligned}\psi_P(x,y,z,w) \equiv &\big[x = 0 \wedge z = 1 \wedge w = 0\big] \vee \big[x = 0 \wedge z = 0 \wedge P(y,w)\big] \vee \\ &\big[x = 1 \wedge z = 0 \wedge y = w \wedge \neg E(y)\big] \vee \big[x = 1 \wedge z = 1 \wedge y = w \wedge E(y)\big].\end{aligned}$$

Análogamente, para $N^{I(\mathcal{A})}$ se tiene la siguiente fórmula:

$$\begin{aligned}\psi_N(x,y,z,w) \equiv &\big[x = 0 \wedge z = 0 \wedge N(y,w)\big] \vee \big[x = 1 \wedge z = 1 \wedge w = 0 \wedge \neg E(y)\big] \vee \\ &\big[x = 1 \wedge z = 1 \wedge y = w \wedge E(y)\big].\end{aligned}$$

El último implicante que aparece tanto en ψ_P como en ψ_N es una condición auxiliar, la cual permite ver cada elemento de $|I(\mathcal{A})|$ como el índice de una cláusula, útil para aquellos casos en los que y no represente una variable universal. Por construcción, $I = \lambda_{xyzw}\langle \psi_0, \psi_E, \psi_P, \psi_N\rangle$ es una proyección de $\exists$UNSAT a $\exists\exists!$SAT. □

Ejemplo 3.3. En [13], Immerman propone una proyección de SAT a 3COLOR (problema de colorabilidad, Ejemplo 2.3) con una propiedad muy favorable.

Sean $I : \text{STRUC}[\sigma_{\text{fnc}}] \to \text{STRUC}[\sigma_g]$ tal reducción, $\mathcal{A}$ una σ_{fnc}-estructura y el grafo $I(\mathcal{A})$ su respectiva interpretación. $I(\mathcal{A})$ es 3-colorable si y sólo si $\mathcal{A}$ es satisfactible y además,

toda coloración de $I(\mathcal{A})$ sólo puede ser inducido por un certificado de $\mathcal{A}$. De esta forma, podemos plantear una reducción de ∃UNSAT a CERTAIN VICTORY, donde los vértices que pertenecen al Jugador 1 son aquellos que dependen de las variables existenciales de $\mathcal{A}$. Este reconocimiento se puede hacer mediante fórmulas proyectivas. □

EJEMPLO 3.4. En [17] se propone una reducción de ∃∀SAT a 2CC[1] que puede adaptarse como proyección. Sea $\mathcal{A}_\phi = \langle [n], E, Q, M\rangle$ una instancia de ∃∀SAT. El subíndice ϕ indica que estamos considerando también la estructura básica de la fórmula Booleana. Se construye un grafo $\mathcal{G}_\phi$ con las siguientes características:

- $\mathcal{G}_\phi$ constará de $6n$ vértices;
- Para cada variable Booleana x_i en ϕ se considerarán cuatro tipo de etiquetas en $\mathcal{G}_\phi$: x_i, $\overline{x_i}$, x_i' y $\overline{x_i}'$;
- Para cada implicante p_i de ϕ se considerarán dos tipos de etiquetas en $\mathcal{G}_\phi$: p_i y p_i'.

Esto es respecto al dominio de $\mathcal{G}_\phi$. En cuanto a las aristas:

- Los vértices x y $\overline{x}$ son adyacentes a x' y $\overline{x}'$ respectivamente;
- Si x es una variable existencial de ϕ, entonces el vértice x' es adyacente a $\overline{x}'$;
- Los vértices p_i y p_i' son adyacentes, al igual que p_i' y p_{i+1}. En otras palabras, la sucesión $p_1 - p_1' - p_2 - p_2' - \cdots - p_n - p_n'$ es un camino en $\mathcal{G}_\phi$;
- Si x no es variable existencial de ϕ, entonces los vértices x' y $\overline{x}'$ son adyacentes a p_n';
- El conjunto de vértices x y $\overline{x}$ induce el mayor grafo que no contiene a las aristas $\{x, \overline{x}\}$;
- El vértice x es adyacente a p, si x aparece en el implicante p de ϕ;
- El vértice $\overline{x}$ es adyacente a p, si $\neg x$ aparece en el implicante p de ϕ;
- Si ninguno de los literales x o $\neg x$ aparecen en el implicante p, entonces los vértices x y $\overline{x}$ son adyacentes a p.

En la FIGURA 3.1 se puede ver con detalles una de tales construcciones. El entorno punteado que rodea a las variables x y $\overline{x}$ representa el grafo inducido por esas variables, como se establece en el quinto inciso de la lista anterior.

[1] En realidad la reducción de Marx parte del problema ∃∀3SAT = ∃∀SAT ∩ 3FND, donde 3FND es el conjunto de formulas Booleanas en FND con a los sumo tres literales por implicante. Este problema también es Σ_2^p-completo [24]. La prueba de la completitud de 2CC no depende del número de literales que hayan en cada implicante, es por ello que decidimos trabajar simplemente con el problema ∃∀SAT.

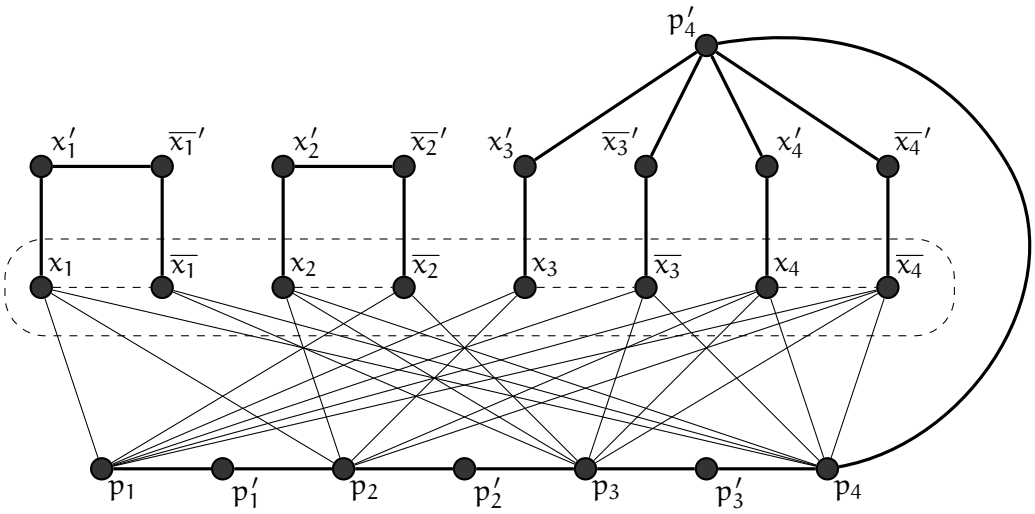

FIGURA 3.1. *Grafo asociado a la fórmula Booleana*
$\phi_1 \equiv (x_1 \wedge \neg x_2) \vee (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_3) \vee (x_2 \wedge \neg x_3)$
con x_1 *y* x_2 *como las variables existenciales .*

La función $I : \mathrm{STRUC}[\sigma'_{\mathrm{fnd}}] \to \mathrm{STRUC}[\sigma_g]$ dada por $I(\mathcal{A}_\phi) = \mathcal{G}_\phi$ es tal que

$$\mathcal{A}_\phi \in \exists\forall\mathrm{SAT} \qquad \Leftrightarrow \qquad \mathcal{G}_\phi \in 2\mathrm{CC}\,.$$

Los detalles de este hecho pueden verse en [17], por lo que sólo nos limitaremos a mostrar que I es una proyección. La aridad de I es $k = 4$. Si $\mathcal{A}_\phi$ tiene dominio $[n]$ entonces el dominio de $\mathcal{G}_\phi$ será

$$|\mathcal{G}_\phi| = \big\{(i,j,k,x) \in [n]^4 : ijk = \mathrm{bin}(m) \text{ para algún } m = 1,\ldots,6\big\}.$$

Aquí $\mathrm{bin}(m)$ es la representación binaria de m en tres bits y la igualdad $ijk = \mathrm{bin}(m)$ es una comparación bit a bit. Todo esto es expresable en primer orden mediante fórmulas numéricas. Denotamos esta fórmula simplemente por

$$\varphi_0(i,j,k,x) \equiv ijk = \mathrm{bin}(1) \vee \cdots \vee ijk = \mathrm{bin}(6).$$

$|\mathcal{G}_\phi|$ tiene exactamente $6n$ elementos. Para cada $x \in [n]$, las tuplas $(0,0,1,x)$, $(0,1,0,x)$, $(0,1,1,x)$ y $(1,0,0,x)$ representan a los vértices x, x', $\overline{x}'$ y $\overline{x}$ respectivamente. Mientras que para la variable $p \in [n]$, $(1,0,1,p)$ y $(1,1,0,p)$ representan a los vértices p y p' respectivamente. Ahora describiremos una fórmula $\varphi_A((i_1,j_1,k_1,x),(i_2,j_2,k_2,p))$ que caracterizará al conjunto de aristas de $\mathcal{G}_\phi$. φ_A será la disyunción de todas las fórmulas que mostremos a continuación.

Los vértices x y $\overline{x}$ son adyacentes a x' y $\overline{x}'$ respectivamente:

$$\big(x = p \wedge i_1j_1k_1 = 001 \wedge i_2j_2k_2 = 010\big) \vee \big(x = p \wedge i_1j_1k_1 = 011 \wedge i_2j_2k_2 = 100\big).$$

Si x es una variable existencial, entonces el vértice x' es adyacente a $\overline{x}'$:

$$\big(E(x) \wedge x = p \wedge i_1j_1k_1 = 010 \wedge i_2j_2k_2 = 011\big).$$

La sucesión $p_1 - p_1' - p_2 - p_2' - \cdots - p_n - p_n'$ es un camino en $\mathcal{G}_\phi$:

$$\big(x = p \wedge i_1j_1k_1 = 101 \wedge i_2j_2k_2 = 110\big) \vee \big(\mathrm{suc}(x, p) \wedge i_1j_1k_1 = 110 \wedge i_2j_2k_2 = 101\big).$$

Si x no es variable existencial, entonces los vértices x' y $\overline{x}'$ son adyacentes a p_n':

$$\big(\neg E(x) \wedge p = \text{máx} \wedge i_1j_1k_1 = 010 \wedge i_2j_2k_2 = 110\big)\vee$$
$$\big(\neg E(x) \wedge p = \text{máx} \wedge i_1j_1k_1 = 011 \wedge i_2j_2k_2 = 110\big).$$

El conjunto de vértices x y $\overline{x}$ induce el mayor grafo que no contiene a las aristas $\{x, \overline{x}\}$:

$$\big(x \neq p \wedge i_1j_1k_1 = i_2j_2k_2 = 001\big) \vee \big(x \neq p \wedge i_1j_1k_1 = i_2j_2k_2 = 100\big)\vee$$
$$\big(x \neq p \wedge i_1j_1k_1 = 001 \wedge i_2j_2k_2 = 100\big).$$

Las últimas tres propiedades se establecen con la siguiente fórmula:

$$\big(i_1j_1k_1 = 001 \wedge i_2j_2k_2 = 101 \wedge \neg M(x, p)\big) \vee \big(i_1j_1k_1 = 100 \wedge i_2j_2k_2 = 101 \wedge \neg Q(x, p)\big).$$

Note que todas estas fórmulas son proyectivas y además las partes numéricas son mutuamente excluyentes. Finalmente, la interpretación $I = \lambda_{ijkx}\langle \varphi_0, \varphi_A\rangle$ es una proyección que reduce el problema $\exists\forall$SAT a 2CC. □

Recopilamos en un teorema los resultados obtenidos en este capítulo. Todas las proyecciones fueron construidas explícitamente por el autor de este trabajo, si bien fueron adaptadas de reducciones de otros tipos ya conocidas.

Teorema 3.2. Los siguientes problemas son Σ_2^p-completos vía proyecciones:

1. $\exists\forall$SAT
2. $\exists$UNSAT
3. $\exists\exists!$SAT
4. CERTAIN VICTORY
5. 2CC

Capítulo 4

Superfluidad

En el capítulo anterior se mostró la completitud en Σ_2^p de varios problemas por la vía tradicional: reducciones. Veremos como resultado principal que es posible demostrar completitud en Σ_2^p utilizando técnicas exclusivamente sintácticas.

Los conceptos y resultados que se mostrarán a continuación fueron desarrollados en [6], motivados por una conjetura planteada en [19].

Conjetura (Medina). Si Φ es una sentencia en $\exists\mathbf{SO}$ y ψ una sentencia en $\mathbf{FO}$, ambas sobre un vocabulario en común σ, entonces la $\mathbf{NP}$-completitud de MOD$[\psi \wedge \Phi]$ implica la $\mathbf{NP}$-completitud de MOD$[\Phi]$.

Recordemos que todo problema de $\mathbf{NP}$ se expresa por medio de una fórmula

$$\Phi \equiv \exists S_1^{a_1} \cdots S_g^{a_g} \forall x_1 \cdots x_k \varphi(x_1, \ldots, x_k), \tag{4.1}$$

donde φ es una fórmula de primer orden (sin cuantificadores) en FNC sobre un vocabulario $\sigma \cup \langle S_1^{a_1}, \ldots, S_g^{a_g} \rangle$. Si en φ hay cláusulas que no dependen de las variables relacionales S_i, entonces podemos reescribir esta fórmula como la conjunción

$$\varphi(x_1, \ldots, x_k) \equiv \varphi_1(x_1, \ldots, x_k) \wedge \varphi_2(x_1, \ldots, x_k),$$

donde φ_1 está definida sobre σ y φ_2 sobre $\sigma \cup \langle S_1^{a_1}, \ldots, S_g^{a_g} \rangle$. De esta manera,

$$\Phi \equiv \Big(\forall x_1 \cdots x_k \varphi_1(x_1, \ldots, x_k)\Big) \wedge \Big(\exists S_1^{a_1} \cdots S_g^{a_g} \forall x_1 \cdots x_k \varphi_2(x_1, \ldots, x_k)\Big).$$

Hemos reescrito Φ como la conjunción de una sentencia universal de primer orden y una sentencia existencial de segundo orden. En función de esto, se puede replantear la conjetura de Medina. En [6] se demuestra que si ésta se restringe al extracto $\forall\mathbf{FO}$ de sentencias universales de primer orden, entonces es válida, y se obtienen resultados similares para las clases de complejidad $\mathbf{NL}$, $\mathbf{P}$ y $\mathbf{coNP}$. El propósito de este capítulo es incluir a Σ_2^p en la lista de clases con tal propiedad.

4.1. Definiciones y Resultado Principal

Los siguientes conceptos y resultados fueron tomados de [6], donde fueron enunciados por primera vez.

DEFINICIÓN 4.1. Sean $\mathcal{L}$ una lógica, $\mathbf{C}$ la clase de complejidad capturada por $\mathcal{L}$ y $\mathcal{L}'$ un fragmento de $\mathcal{L}$.

1. ψ es *superflua con respecto a* $\mathcal{L}$ si para toda sentencia $\Phi \in \mathcal{L}$ la $\mathbf{C}$-completitud de $\mathrm{MOD}[\psi \wedge \Phi]$ implica la $\mathbf{C}$-completitud de $\mathrm{MOD}[\Phi]$.
2. $\mathcal{L}'$ es *superflua con respecto a* $\mathcal{L}$ si toda sentencia en $\mathcal{L}'$ es superflua con respecto a $\mathcal{L}$.

Bajo el contexto de esta definición, lo dicho en el preámbulo se resume a

PROPOSICIÓN 4.1. $\forall\mathbf{FO}$ es superflua con respecto a **NL**, **P**, **NP** y **coNP**.

Y lo que queremos probar es

PROPOSICIÓN 4.2. $\forall\mathbf{FO}$ es superflua con respecto a Σ_2^p.

Este resultado será consecuencia de una propiedad más fuerte que expondremos en su debido momento. Antes mostraremos varios conceptos necesarios.

DEFINICIÓN 4.2. Sean $\varphi(x_1, \ldots, x_r) \in \mathbf{FO}[\sigma]$ una fórmula, $u_1, \ldots, u_r \in [n]$ y $S \subseteq \mathrm{STRUC}[\sigma]$.

1. $\langle \varphi(\overline{x}), \overline{u} \rangle$ es n-*consistente* si existe $\mathcal{A} \in \mathrm{STRUC}[\sigma]$ tal que $\|\mathcal{A}\| = n$ y $\mathcal{A} \models \varphi(\overline{u})$.
2. $\langle \varphi(\overline{x}), \overline{u} \rangle$ es n-*consistente en* S si existe $\mathcal{A} \in S$ tal que $\|\mathcal{A}\| = n$ y $\mathcal{A} \models \varphi(\overline{u})$.

Si no hay riesgo de confusión, se escribirá "$\varphi(\overline{u})$ es n-consistente (en S)", en vez de la notación dada en la definición anterior.

DEFINICIÓN 4.3. Sean $\sigma = \langle R_1^{a_1}, \ldots, R_s^{a_s}, c_1, \ldots, c_t \rangle$ un vocabulario, n y r dos números naturales. Sea $S \subseteq \mathrm{STRUC}[\sigma]$.

1. S es $(n, 0)$-*uniforme* si para todo $m \in \mathbb{N}$ con $m \geq n$ y toda sucesión $b_1, \ldots, b_t \in [m]$ existe $\mathcal{A}_m \in S$ con $\|\mathcal{A}_m\| = m$ y tal que $\mathcal{A}_m \models (c_1 = b_1) \wedge \cdots \wedge (c_t = b_t)$.
2. S es (n, r)-*uniforme* si para todo $m \in \mathbb{N}$ con $m \geq n$, toda sucesión de σ-literales $L_1, \ldots, L_r$ con L_j igual a $R_{i_j}(\overline{x})$ o $\neg R_{i_j}(\overline{x})$, toda sucesión de tuplas $\overline{u}_1, \ldots, \overline{u}_r$ con $\overline{u}_j \in [m]^{a_{i_j}}$ y toda sucesión $b_1, \ldots, b_t \in [m]$, la m-consistencia de

$$\varphi(\overline{u}, \overline{b}) \equiv \bigwedge L_j(\overline{u}_j) \wedge \bigwedge (c_k = b_k)$$

implica su m-consistencia en S (es decir, si existen modelos de $\varphi(\overline{u}, \overline{b})$ de tamaño m, entonces al menos uno pertenece a S).

Como resultado principal de este trabajo (así como ejemplo de las últimas definiciones), consideremos la siguiente proposición. Muchas de las afirmaciones dependen de ciertas convenciones establecidas en el Ejemplo 1.3.

Proposición 4.3. 2CC es $(2k+1, k)$-uniforme para todo $k \geq 1$.

Demostración. Requerimos del vocabulario $\sigma_g = \langle A^2 \rangle$. Veamos primero el caso $k = 1$ para hacernos una idea de cómo proceder en general. Sean $m \geq 3$ y el literal $L(u, v)$, donde L es A o $\neg A$ y $u, v \in [m]$. Note que si L es A, la fórmula $L(u, v)$ es consistente si $u \neq v$, pues siempre podremos construir un grafo (sin lazos) tal que $\{u, v\}$ es arista del grafo. Los siguientes análisis son hechos bajo la suposición de que $u \neq v$.

Supongamos que L es A y sea $\mathcal{G} \in \text{STRUC}[\sigma_g]$ el grafo completo sobre el dominio $[m]$. $\mathcal{G}$ satisface $A(u, v)$ y además es una instancia positiva de 2CC, pues el único clique maximal de $\mathcal{G}$ es él mismo y por lo tanto basta tomar una 2-coloración que asigne colores distintos a un par de vértices cualesquiera de $\mathcal{G}$.

Si L es $\neg A$, sea $\mathcal{G} \in \text{STRUC}[\sigma_g]$ el grafo sobre el dominio $[m]$, que posee todas las aristas posibles salvo $\{u, v\}$. $\mathcal{G}$ satisface $\neg A(u, v)$ y además es también instancia positiva de 2CC. En efecto, los únicos cliques maximales de $\mathcal{G}$ son los subgrafos inducidos por los conjuntos $[m] \setminus \{u\}$ y $[m] \setminus \{v\}$. La coloración sobre $\mathcal{G}$ que asigna el color verde a u y a v, y al resto de vértices el color rojo, es un certificado de $\mathcal{G}$.

Hasta aquí hemos demostrado que 2CC es $(3, 1)$-uniforme. El caso general se demuestra con argumentos muy similares. Sea k un número natural y $m \geq 2k + 1$. Consideremos las condiciones

$$L_1(u_1, v_1), L_2(u_2, v_2), \ldots, L_k(u_k, v_k),$$

donde para todo índice i, L_i es el predicado A o $\neg A$. Además de pedir la condición sin lazos, estas k fórmulas son m-consistentes siempre que no haya un par (u, v) e índices $i \neq j$ tales que $L_i(u, v)$ con $L_i = A$ y $L_j(u, v)$ con $L_j = \neg A$. Los siguientes análisis son hechos bajo la suposición de que las k fórmulas son consistentes.

Supongamos primero que las k condiciones son de la forma

$$A(u_1, v_1), A(u_2, v_2), \ldots, A(u_k, v_k),$$

es decir, cuando todos los L_i son A. Definiendo $\mathcal{G}$ como el grafo completo sobre el dominio $[m]$, se obtiene lo requerido.

Supongamos ahora que ciertos L_i son $\neg A$. Sin ningún tipo de impedimento los arreglamos de la siguiente manera:

$$A(u_1, v_1), \ldots, A(u_j, v_j), \neg A(u_{j+1}, v_{j+1}), \ldots, \neg A(u_k, v_k),$$

para algún $j \in \{0, \ldots, k-1\}$ (cuando $j = 0$, quiere decir que todos los literales son negativos). Definamos el grafo $\mathcal{G} = \langle [m], A^{\mathcal{G}} \rangle$, donde $A^{\mathcal{G}}$ es el conjunto de todas las posibles aristas sobre $[m]$ salvo $\{u_i, v_i\}$ con $i > j$. Este grafo satisface las k condiciones y además es instancia positiva de 2CC. En efecto, sea el conjunto

$$R = \{x \in [m] : x \notin \{u_i, v_i\} \text{ para todo } i > j\},$$

el cual es no vacío para todo $m \geq 2k+1$, pues en el peor de los casos (cuando $j = 0$), sólo se pueden abarcar hasta $2k$ vértices distintos con las k condiciones que se proponen. R tampoco es $[m]$ pues partimos de la suposición de que hay literales negativos.

Este conjunto R tiene la propiedad de que si $x \in R$ y $u \notin R$, entonces $\{x, u\}$ es arista del grafo $\mathcal{G}$. Además, el subgrafo de $\mathcal{G}$ inducido por R es un clique, pero no es maximal, pues por la observación anterior, si $u \notin R$, entonces $R \cup \{u\}$ también induce un clique.

Recíprocamente, si $w_1, \ldots, w_s \notin R$ forman un clique, éste se puede extender a un clique más grande anexándole algún elemento de R, más aún, $R \cup \{w_1, \ldots, w_s\}$ también induce un clique.

Consecuentemente, todo clique maximal de $\mathcal{G}$ debe tener necesariamente un vértice en R y otro vértice que no está en R, en otras palabras, R es la 2-coloración de $\mathcal{G}$ que lo certifica como instancia positiva de 2CC. □

El resultado anterior es importante, ya que nos proporcionará una familia numerable de problemas en Σ_2^p con características especiales.

DEFINICIÓN 4.4. Sea $\mathbf{C}$ una clase de complejidad. Una familia $\mathcal{F}$ de problemas definidos sobre un vocabulario σ es:

1. $\mathbf{C}$-*completa* si cada problema en $\mathcal{F}$ es $\mathbf{C}$-completo vía proyecciones.

2. *Uniforme* si para todo $k > 0$, existe un número natural n_k y un problema $S_{n_k} \in \mathcal{F}$ tal que S_{n_k} es (n_k, k)-uniforme y contiene a todas las σ-estructuras de tamaño menor o igual que n_k.

El resultado principal de [6] establece que

TEOREMA 4.4. Sea $\mathbf{C}$ una clase de complejidad capturada por la lógica $\mathcal{L}$. Si $\mathbf{C}$ contiene una familia completa y uniforme $\mathcal{F}$ entonces $\forall\mathbf{FO}$ es superfluo con respecto a $\mathbf{C}$.

La siguiente definición y los siguientes lemas (ambos demostrados en [6]) nos permitirán definir una familia $\mathcal{F}$ adecuada en Σ_2^p.

DEFINICIÓN 4.5. Si S es un problema sobre un vocabulario σ, se define para cada número natural n el problema

$$S_n := S \cup \{\mathcal{A} \in \mathrm{STRUC}[\sigma] : \|\mathcal{A}\| < n\},$$

y la familia de problemas

$$\mathcal{F}(S) = \{S_n\}_{n \geq 2}.$$

LEMA 4.5. Sean S y T un par de problemas. Si $S \subseteq T$ y S es (n, r)-uniforme entonces T también es (n, r)-uniforme.

LEMA 4.6. Sean $\mathcal{L}$ una lógica que contiene a $\mathbf{FO}$ y que captura a una clase de complejidad $\mathbf{C}$. Si S es un problema en $\mathbf{C}$ entonces S_n pertenece a $\mathbf{C}$ para cada número natural n.

PROPOSICIÓN 4.7. $\mathcal{F}(2\mathrm{CC})$ es una familia completa uniforme en Σ_2^p.

DEMOSTRACIÓN. La uniformidad de $\mathcal{F}(2\mathrm{CC})$ es consecuencia de la PROPOSICIÓN 4.3 y del LEMA 4.5. La contención de $\mathcal{F}(2\mathrm{CC})$ en Σ_2^p se debe al LEMA 4.6. Sólo nos falta demostrar que $(2\mathrm{CC})_n$ es Σ_2^p-completo para cada n. Basta con definir una proyección ρ de 2CC a $(2\mathrm{CC})_n$.

Para cada n, existe un número entero k_n tal que $2k_n > n$. Sea $\mathcal{G}$ una σ_g-estructura. Definamos el grafo $\rho(\mathcal{G})$ compuesto por k_n copias disjuntas de $\mathcal{G}$. Como toda estructura se asume con al menos dos elementos distintos en su dominio, entonces el dominio de $\rho(\mathcal{G})$ es siempre mayor que n para toda σ_g-estructura $\mathcal{G}$. Por lo tanto, $\mathcal{G}$ es instancia positiva de 2CC si y sólo si $\rho(\mathcal{G})$ es instancia positiva de $(2\mathrm{CC})_n$.

Finalmente, ρ es una proyección de aridad $\log(k_n) + 1$, donde las primeras $\log(k_n)$ componentes de las tuplas $\overline{x}$ se utilizan para identificar bit a bit alguna de las copias del grafo, y la última componente para etiquetar los vértices de cada copia. □

Por el Teorema 4.4 y la última proposición, queda demostrada la Proposición 4.2. A continuación veremos cómo utilizar esta nueva propiedad de Σ_2^p para demostrar completitud en esta clase.

4.2. Aplicaciones

Proposición 4.8. VC SAT es Σ_2^p-completo.

Demostración. Sea el vocabulario $\sigma_{fnd} \cup \langle V^2, K^1 \rangle$ y la fórmula de segundo orden Ψ_{vc}, ambos del Ejemplo 2.5. Consideremos la sentencia de primer orden

$$\varphi \equiv \forall xk\Big[\big((V(x,k) \leftrightarrow k = 0) \oplus V(x,k)\big) \wedge \big(K(k) \leftrightarrow bit(máx, k)\big)\Big].$$

Sea $\mathcal{A} \in \text{MOD}[\varphi \wedge \Psi_{vc}]$. Esta estructura, además de ser instancia positiva de VC SAT, es aquella tal que su costo se define como el valor máximo de su dominio y los posibles valores para cada una de sus variables son 1 o un valor que sobrepasa el costo. Definamos entonces el conjunto

$$E^{\mathcal{A}} = \{x \in |\mathcal{A}| : \nu(x) = 1\}.$$

El conjunto E es aquel para el cual se satisface Ψ_{sat}, como subfórmula de Ψ_{vc}. Ahora, sea $\mathcal{A}$ una instancia de $\exists\forall$SAT con dominio n. Si definimos los valores de las variables Booleanas como 1 si es existencial, $2^n - 1$ si es universal y el costo como el valor n, entonces $\mathcal{A}$ es instancia positiva de $\exists\forall$SAT sólo si satisface la fórmula $\varphi \wedge \Psi_{vc}$ con estas interpretaciones de valores y costos. Note además que todo esto se puede realizar mediante fórmulas de primer orden proyectivas.

En otras palabras, $\text{MOD}[\varphi \wedge \Psi_{vc}]$ y $\exists\forall$SAT son esencialmente el mismo problema. Como $\exists\forall$SAT es Σ_2^p-completo, entonces $\text{MOD}[\varphi \wedge \Psi_{vc}]$ también lo es y por la Proposición 4.2, $\text{MOD}[\Psi_{vc}] = $ VC SAT también es Σ_2^p-completo. □

Otros resultados similares se resumen en la siguiente:

Proposición 4.9. Las versiones de valor y costo de los problemas $\exists$UNSAT, $\exists\exists!$SAT y CERTAIN VICTORY son Σ_2^p-completos.

El problema $\exists\exists!$SAT se puede adaptar a una propiedad muy similar, considerando ahora una asignación de verdad fija para las variables no existenciales. Llamamos a este problema $\exists\exists!_*$SAT, donde la fórmula de segundo orden que lo describe es parecida a (2.3) salvo que T ya no se considera una variable relacional, sino un símbolo más del vocabulario subyacente al problema. Originalmente, $\exists\exists!_*$SAT fue estudiado en [**11**] como un caso particular de problemas en el que los certificados se pueden definir parcialmente.

PROPOSICIÓN 4.10. $\exists\exists!_*$SAT es Σ_2^p-completo.

DEMOSTRACIÓN. Consideremos el vocabulario $\sigma = \sigma'_{fnc} \cup \langle T^1 \rangle$ y Ψ en $\exists\forall\mathbf{SO}[\sigma]$ tal que $\mathrm{MOD}[\Psi] = \exists\exists!_*$SAT. Sea la sentencia universal de primer orden

$$\varphi \equiv \forall x \big[T(x) \leftrightarrow \neg E(x) \big].$$

Una estructura $\mathcal{B}$ en $\mathrm{MOD}[\varphi \wedge \Psi]$ es una instancia positiva de $\exists\exists!_*$SAT, con valor asignado **true** para cada variable no existencial. En el EJEMPLO 3.2 vimos que un certificado de la interpretación $I(\mathcal{A})$, de una fórmula $\mathcal{A}$ en FNC, sólo podía extenderse a la asignación T dada por la fórmula φ. Es decir, $I(\mathcal{A}) \in \exists\exists!$SAT si y sólo si $\langle I(\mathcal{A}), T \rangle \in \exists\exists!_*$SAT. De aquí se obtiene que $\mathrm{MOD}[\varphi \wedge \Psi]$ es Σ_2^p-completo, y por lo tanto, $\mathrm{MOD}[\Psi] = \exists\exists!_*$SAT es también Σ_2^p-completo. □

4.3. Superfluidad en Π_2^p

A diferencia de lo que sucede con propiedades como pertenencia, reducibilidad o completitud, no podemos concluir por dualidad que $\forall\mathbf{FO}$ es superflua respecto a Π_2^p. El problema $(2CC)^c$ es Π_2^k-completo, pero no es $(3,1)$-uniforme, pues todos los grafos de tamaño 3 satisfacen la propiedad 2CC. Tampoco es fácil determinar si es $(2k+1, k)$-uniforme para $k > 1$. Sin embargo, tenemos la libertad de escoger una nueva secuencia de valores.

PROPOSICIÓN 4.11. $(2CC)^c$ es $(2k+5, k)$-uniforme para todo $k \geq 1$.

DEMOSTRACIÓN. Sean las condiciones $L_1(u_1, v_1), \ldots, L_k(u_k, v_k)$, donde L_i es el predicado A o $\neg A$. Supongamos que estas k condiciones son m-consistentes con $m \geq 2k+5$ y consideremos el menor de los grafos que modela las condiciones, es decir, $\mathcal{G} = \langle [m], A^{\mathcal{G}} \rangle$ donde

$$A^{\mathcal{G}} = \big\{ (u,v) \in [m]^2 : (u,v) = (u_i, v_i) \text{ para algún } i = 1, \ldots, k \text{ y } L_i = A \big\}.$$

Note que como a lo sumo hay $2k$ vértices de $\mathcal{G}$ afectados por las k condiciones, entonces siempre habrán al menos cinco vértices independientes (vértices que no están conectados a ningún otro). Con estos cinco vértices, digamos $w_1, \ldots, w_5$, construimos un ciclo y definimos un nuevo grafo $\mathcal{G}' = \langle [m], A^{\mathcal{G}'} \rangle$ donde

$$A^{\mathcal{G}'} = A^{\mathcal{G}} \cup \{(w_1, w_2), \ldots, (w_5, w_1)\}.$$

$\mathcal{G}'$ es un grafo disconexo que consta de un subgrafo conexo maximal (no hay otro subgrafo conexo de $\mathcal{G}'$ que lo contenga) que no satisface la propiedad 2CC (Ejemplo 2.4). Por lo tanto, $\mathcal{G}'$ es instancia positiva de $(2CC)^c$. □

Utilizando los Lemas 4.5 y 4.6 y la misma reducción de la Proposición 4.7 se concluyen las siguientes:

Proposición 4.12. $\mathcal{F}((2CC)^c)$ es una familia completa uniforme en Π_2^p.

Proposición 4.13. $\forall$**FO** es superflua con respecto a Π_2^p.

Capítulo 5

Conclusiones y futuras investigaciones

En este trabajo se trata con profundidad el tema de completitud vía proyecciones en las clases de complejidad Σ_2^p y Π_2^p, lo cual difiere de la mayoría de trabajos que se centran en estas clases, donde suelen utilizar otro tipo de reducciones. En el Teorema 3.2 se listan aquellos problemas que se demuestran completos vía proyecciones, haciendo una construcción explícita de las reducciones proyectivas mostradas en los ejemplos previos. También se estudian los problemas desde un punto de vista netamente sintáctico al utilizar recurrentemente el Teorema 2.3.

La Proposición 4.1, demostrada en [6], se extiende al anexar tanto Σ_2^p como Π_2^p en la lista de aquellas clases donde se puede aplicar la propiedad de superfluidad, esto como consecuencia de las Proposiciones 4.3, 4.7, 4.11 y 4.12. Además, utilizando estos resultados se demuestra la Σ_2^p-completitud de varios problemas numéricos como VC SAT, así como cualquier otro problema Σ_2^p-completo que admita una versión de valor y costo, un tipo de problemas definidos formalmente en [4]. Todos las proposiciones demostradas en la primera sección del Capítulo 4 se plantearon con el propósito específico de aplicar la técnica de superfluidad por primera vez en Σ_2^p.

Siguiendo la línea de investigación de [6] y de este trabajo, se espera obtener resultados similares en los niveles superiores de la Jerarquía Polinomial. La simplicidad con que se expresa el problema 2CC fue lo que permitió obtener en Σ_2^p varias propiedades interesantes como la construcción de una familia completa y uniforme. En principio, cualquier problema Σ_k^p-completo (vía proyección) es candidato para poder construir en Σ_k^p una familia completa y uniforme, pero esta última propiedad puede resultar difícil de verificar según las características estructurales que definan al problema. Un posible método podría ser reinterpretar 2CC a una versión completa en cada nivel de la jerarquía polinomial y asegurar mediante un razonamiento inductivo la uniformidad de estos problemas. Tampoco se ha intentado aplicar este proceso con problemas genéricos ya conocidos como la versión cuantificada de SAT o la versión en k pasos de CERTAIN VICTORY.

La conjetura de Medina, mencionada al comienzo del Capítulo 4, sigue siendo de interés para futuras investigaciones, puesto que aún no se sabe si existen otros fragmentos de primer orden con las mismas características de superfluidad que $\forall\mathbf{FO}$.

Hasta ahora el concepto de uniformidad parece estar arraigado a cualquier clase de complejidad con algún problema completo vía proyecciones. Podría estudiarse si esta propiedad no es aplicable en alguna clase de complejidad poco usual, por ejemplo

$$\mathbf{DP} = \{L_1 \cap L_2 : L_1 \in \mathbf{NP} \text{ y } L_2 \in \mathbf{coNP}\},$$

lo que representaría un avance en el estudio de clases de complejidad estrictamente contenidas en otras.

Apéndice A

Problemas referidos en el texto

Aquí se recogen las definiciones de todos los problemas mencionados a lo largo del trabajo, usando el formato instancia-pregunta. La mayoría de las referencias fueron tomadas del compendio de Schaefer y Umans [23], donde a su vez citan la fuente original del problema.

1. 2CC a.k.a. 2 CLIQUE COLORING

   Instancia: Grafo $\mathcal{G} = \langle V, A \rangle$.

   Pregunta: ¿Existe una 2-coloración sobre V tal que ningún clique maximal de $\mathcal{G}$ sea monocromático?
2. CERTAIN VICTORY

   Instancia: Grafo $\mathcal{G} = \langle V, A \rangle$ y un subconjunto $U \subset V$.

   Pregunta: ¿Existe una 3-coloración de U que no pueda extenderse a una 3-coloración de $\mathcal{G}$?
3. 3COLOR

   Instancia: Grafo $\mathcal{G} = \langle V, A \rangle$.

   Pregunta: ¿Existe una 3-coloración de $\mathcal{G}$?
4. SAT

   Instancia: Fórmula Booleana ϕ en FNC.

   Pregunta: ¿Es ϕ satisfactible?
5. $\exists\forall$SAT

   Instancia: Fórmula Booleana ϕ en FND, con variables clasificadas distintivamente como existenciales y universales.

   Pregunta: ¿Existe una asignación de valores de las variables existenciales tal que ϕ se satisface para toda asignación de valores de las variables universales?
6. $\exists!$SAT

   Instancia: Fórmula Booleana ϕ en FNC.

   Pregunta: ¿Es ϕ satisfactible y con un único certificado?

7. $\exists\exists!$SAT

   Instancia: Fórmula Booleana ϕ en FNC, con algunas de sus variables clasificadas como existenciales.

   Pregunta: ¿Existe una asignación de valores de las variables existenciales que se pueda extender a un único certificado de ϕ?

8. $\exists\exists!_*$SAT

   Instancia: Fórmula Booleana ϕ en FNC, con algunas de sus variables clasificadas como existenciales y una asignación de las variables no existenciales.

   Pregunta: ¿Existe una única extiención de la asignación dada que sea un certificado de ϕ?

9. UNSAT

   Instancia: Fórmula Booleana ϕ en FNC.

   Pregunta: ¿Es ϕ insatisfactible?

10. $\exists$UNSAT

    Instancia: Fórmula Booleana ϕ en FNC, con variables clasificadas distintivamente como existenciales y universales.

    Pregunta: ¿Existe una asignación de valores de las variables existenciales tal que ϕ no se satisface para ninguna asignación de valores de las variables universales?

11. VALID

    Instancia: Fórmula Booleana ϕ en FND.

    Pregunta: ¿Es ϕ una tautología?

12. VC SAT

    Instancia: $\mathcal{A} = \langle [n], Q, M\rangle$ una σ_{fnd}-estructura, un valor $\nu(x)$ para cada variable Booleana x, y un costo K.

    Pregunta: ¿Existe un subconjunto E de variables tal que la suma de todos los valores de elementos en E no sea mayor que K y de modo que $\mathcal{A} \cup \langle E\rangle$ sea una instancia positiva de $\exists\forall$SAT?

Bibliografía

[1] E. Allender, J. Balcázar, N. Immerman, *A First-order Isomorphism Theorem*. SIAM J. Comput. 26(2), pp. 557-567. 1997. 1

[2] S. Arora, B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, New York. 2009. 3, 24

[3] A. Arratia, *Introducción a la Teoría Descriptiva de la Computabilidad*. Escuela Matemática Venezolana, Caracas. 2000. 16, 17, 31

[4] J. Berit, *New Classes of Complete Problems for the Second Level of the Polynomial Hierarchy*. Doctoral Thesis. Technischen Universität Berlin. 2011. 48, 66

[5] N. Borges, B. Bonet, *On Canonical Forms of Complete Problems via First-order Projections*. Cornell University Library, Logic and Computation Complexity. 2007 1

[6] N. Borges, *Técnicas Sintácticas y Combinatorias en el Estudio de la Complejidad Computacional*. Tesis Doctoral. Universidad Simón Bolívar. 2011. 19, 58, 59, 62, 66

[7] A. Chandra, D. Kozen, L. Stockmeyer, *Alternation*. JACM 28, No. 1, pp. 114-133. 1981. 38

[8] S. Cook, *The Complexity of Theorem Proven Procedures*. Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151-158. 1971. 1, 32

[9] R. Fagin, *Generalized First-Order Spectra and Polynomial-Time Recognizable Sets*. Complexity of Computation, ed. R. Karp, SIAM-AMS Proceedings 7, pp. 27-41. 1974. 1

[10] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman and Company. 1979. 1, 19

[11] H. Hatami, H. Manserrat, *On the Computational Complexity of Defining Sets*. Discrete Appl. Math., 149(1-3):101-110, 2005. 64

[12] N. Immerman, *Languages Which Capture Complexity Classes*. 15th ACM STOC Symposium, pp. 347-354. 1983. 31

[13] N. Immerman, *Descriptive Complexity*. New York: Springer-Verlag. 1999. 3, 40, 54

[14] N. Immerman, S. Landau, *The Complexity of Iterated Multiplication*. Information and Computation 116, pp. 103-116. 1995. 33

[15] N. Immerman, *Nondeterministic Space is Closed Under Complementation*. SIAM J. Comput. 17(5), pp. 935-938. 1988. 26

[16] R. Karp, *Reducibility among combinatorial problems*. Complexity of Computer Computations, Plenum Press, New York, pp. 85-103. 1972. 19

[17] D. Marx, *Complexity of clique coloring and related problems*. Manuscript. 2004. 55, 56

[18] J. Medina, N. Immerman, *A Syntactic Characterization of NP-completeness*. IEEE Symposium on Logic in Computer Science, pp. 241-250. 1994. 1, 33

[19] J. Medina, *A Descriptive Approach To The Class NP*. PhD thesis, University of Massachusetts, Amherst, 1997. 58

[20] J. Mosterín, *Lógica de Primer Orden*. Editorial Ariel, 2a. edición. 1976. 3, 8

[21] C. Papadimitriou, *Computational Complexity*. Addison-Wesley. 1994. 3, 16, 17, 26

[22] C. Papadimitriou, M. Yannakakis, *The Complexity of Facets*. ACM Symposium of Theory of Computing. 1982. 45

[23] M. Schaefer, C. Umans, *Completeness in the Polynomial-Time Hierarchy: a compendium*. SIGACT News 33, pp. 32-49. 2002. 19, 43, 68

[24] L. Stockmeyer, *The polynomial-time hierarchy*. Theoretical Computer Science, vol.3, pp. 1-22, 1976. 1, 35, 55