Universidad de La Habana
Facultad de Matemática y Computación

Título de la tesis

Autor:
Raudel Alejandro Gómez Molina

Tutores:
Mcs. Fernando Rodríguez Flores

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Fecha

https://github.com/raudel25/my-thesis

Dedicación akkakkakkakak

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

Resumen en español

Abstract

Resumen en inglés

Índice general

Índice de figuras

Introducción

El problema de satisfacibilidad booleana (SAT) es uno de los problemas más estudiados en la teoría de la computación y la lógica. Consiste en determinar si existe una asignación de valores verdaderos o falsos que satisfaga una fórmula booleana dada, compuesta por variables y operadores lógicos como conjunciones, disyunciones y negaciones. SAT surge en 1971 como el primer problema NP-completo demostrado por Stephen Cook, lo que significa que, en el peor de los casos, su resolución requiere tiempo exponencial respecto al tamaño de la entrada, pero también que muchos otros problemas pueden reducirse a él. Esto implica un especial interés por parte de la comunidad científica en la búsqueda de métodos eficientes para la solución del SAT.

La teoría de lenguajes es una rama fundamental de la Ciencia de la Computación y la matemática que se enfoca en el estudio de los lenguajes formales. Estos lenguajes, definidos a través de gramáticas, autómatas y expresiones regulares, permiten modelar y analizar la estructura de los lenguajes naturales y artificiales. Su aplicación es amplia y abarca desde el diseño de compiladores y procesadores de lenguaje natural hasta la verificación de sistemas y la teoría de la computabilidad.

Los lenguajes formales se clasifican en jerarquías, como la jerarquía de Chomsky, que los organiza según su complejidad y poder expresivo. Esta teoría proporciona las bases para entender cómo se pueden reconocer, generar y transformar cadenas de símbolos, lo que resulta esencial en el desarrollo de herramientas computacionales para el procesamiento de información. Además, la teoría de lenguajes constituye la base de los problemas de la Ciencia de la Computación, ya que cualquier problema puede ser interpretado como un problema de la teoría de lenguajes.

En este trabajo se vinculan las dos ramas de la computación descritas anteriormente, presentando un enfoque para resolver el SAT utilizando formalismos de teoría de lenguajes. Dicho enfoque resulta un tema no evidenciado en la literatura consultada y permite demostrar que una serie de problemas relacionados a la teoría de lenguajes pertenecen a la clase NP-completo.

En estudios anteriores, que siguen la idea presentada en esta investigación, se han mostrado estrategias para la solución de instancias específicas del SAT, usando formalismos de teoría de lenguajes, lo que constituye una solución limitada en su alcance. En cambio, en este trabajo se presenta una alternativa que resuelve cualquier instancia del mismo, lo que, a criterio del autor, resulta una solución cualitativamente superior. Esta sigue siendo una estrategia no eficiente, pero que muestra un nuevo enfoque para resolver el SAT de forma general, y permite abrir nuevas líneas de investigación en este tema.

Para resolver cualquier instancia de SAT empleando formalismos de teoría de lenguajes se propone definir una codificación de una fórmula booleana en una cadena que se pueda interpretar por algún formalismo de la teoría de lenguajes y usando dicha codificación se define el lenguaje de todas las fórmulas booleanas satisfacibles. Entonces si se desea determinar si una fórmula booleana es satisfacible es necesario determinar si la cadena asociada a la fórmula booleana pertenece o no al lenguaje de todas las fórmulas booleanas satisfacibles.

Para construir el lenguaje de las fórmulas booleanas satisfacibles se propone utilizar 2 métodos: el primero utiliza un transductor finito y el segundo utiliza una gramática de concatenación de rango.

A partir de lo expuesto anteriormente se formula como objetivo general de de este trabajo: definir y construir el lenguaje de todas las fórmulas booleanas satisfacibles.

Para cumplir el objetivo general se definen los siguientes objetivos específicos:

- Estudiar el estado del arte referido a los formalismos de teoría de lenguajes y el SAT.
- Establecer una representación del SAT como una cadena que pueda ser interpretada por un formalismo de la teoría de lenguajes.
- Definir el lenguaje de todas las fórmulas booleanas satisfacibles.
- Construir el lenguaje de todas las fórmulas booleanas satisfacibles utilizando un transductor finito.
- Construir el lenguaje de todas las fórmulas booleanas satisfacibles utilizando gramáticas de concatenación de rango.

Para dar cumplimiento a los objetivos trazados, el trabajo se ha estructurado en 4 capítulos: en los 2 primeros se presentan los principales conceptos y definiciones que serán utilizados en el resto de la investigación y en los restantes 2 capítulos se define y construye el lenguaje de todas las fórmulas booleanas satisfacibles.

En el capítulo 1 se presentan los principales conceptos y definiciones de la teoría de lenguajes y el SAT, los cuales son necesarios para la comprensión de los restantes capítulos. Además, se realiza un análisis de 2 trabajos anteriores que muestran cómo solucionar instancias específicas del SAT utilizando un algoritmo polinomial.

En el capítulo 2 se realiza un análisis detallado de las gramáticas de concatenación de rango, presentando las principales definiciones, proceso de derivación y análisis de la complejidad del algoritmo de reconocimiento.

En el capítulo 3 se muestra cómo codificar una fórmula booleana mediante una cadena de símbolos y luego se analiza cómo interpretar una cadena como la asignación de valores para las variables de una fórmula booleana. Posteriormente, se define el lenguaje de todas las fórmulas booleanas satisfacibles y se muestra cómo construir dicho lenguaje mediante un transductor finito. Para finalizar, se demuestra que el problema de la palabra, para todos los formalismo que cumplan ciertas propiedades, las cuales se definen en el capítulo 3, es NP-Duro.

En el capítulo 4 se demuestra que no es necesario construir el lenguaje de todas las fórmulas booleanas satisfacibles mediante transducción finita, ya que existe una gramática de concatenación de rango que reconoce este lenguaje. Por otro lado, se demuestra que las gramáticas de concatenación de rango cubren todos los problemas de la clase NP-Completo.

Capítulo 1

Preliminares

En este capítulo se presentan las principales definiciones y conceptos que serán usados en el resto del trabajo, además se analizan los trabajos anteriores que exponen estrategias de solución del problema de la satisfacibilidad booleana mediante formalismos de la teoría de lenguajes.

1.1. Teoría de Lenguajes

En esta sección se presentan los principales conceptos de teoría de lenguajes que sirven de base al contenido de los capítulos y secciones posteriores, ya que en este trabajo se presentan estrategias para la solución del problema de la satisfacibilidad empleando mecanismos de la teoría de lenguajes.

1.1.1. Conceptos básicos, operaciones y problemas relacionados con lenguajes

Los conceptos básicos de la teoría de lenguajes formales son alfabeto, cadena y lenguaje. Un alfabeto, denotado como Σ, es un conjunto finito y no vacío de símbolos; una cadena es una sucesión finita de símbolos del alfabeto y un lenguaje es un conjunto de cadenas definido sobre un alfabeto. Por ejemplo, el alfabeto $\Sigma = \{1, 0\}$ está formado por los símbolos 0 y 1, 11 y 101 son cadenas sobre el alfabeto Σ y un ejemplo de lenguaje es el conjunto de cadenas de 0 y 1 que terminan en 1.

Existen varias operaciones que pueden realizarse usando alfabetos, cadenas y lenguajes. Seguidamente se presentan algunas de ellas.

Como los lenguajes son conjuntos, todas las operaciones sobre conjuntos también se definen para lenguajes: unión, intersección, complemento [8]. La siguiente operación, homomorfismo, permite definir los transductores finitos usados en el capítulo 3 para construir el lenguaje de todas las fórmulas booleanas satisfacibles.

Definición 1.1. *Dados un alfabeto Σ y un alfabeto Γ, un* ***homomorfismo*** *es una función:*

$$h : \Sigma \to \Gamma^*$$

tal que:

1. *Para cada $a \in \Sigma$, $h(a)$ es una cadena en Γ^*.*

2. *Si $w = a_1a_2\ldots a_n$ es una cadena entonces*

$$h(w) = h(a_1)h(a_2)\ldots h(a_n).$$

Por ejemplo si sobre el alfabeto $\Sigma = \{a,b\}$, se define el homomorfismo h, tal que $h(a) = 0$ y $h(b) = 11$, entonces $h(ab) = 011$.

Una vez que se tiene definido un lenguaje es posible responder preguntas como si una cadena dada pertenece al lenguaje, o si el lenguaje es vacío o contiene algún elemento. Estos problemas se usan en los capítulos 3 y **??** para determinar si una fórmula booleana es satisfacible y en los trabajos [1] y [9] para determinar si una fórmula booleana con ciertas restricciones es satisfacible o no.

Definición 1.2. *El* ***problema de la palabra*** *consiste en determinar si una cadena pertenece a un lenguaje dado.*

Por ejemplo, dado el lenguaje $L = \{w \,|\, \mathrm{last}(w) = 0\}$, determinar si 1100100 pertenece a L.

Todo problema en Ciencia de la Computación se puede reducir a un problema de la palabra, ya que cualquier problema se puede codificar como un lenguaje formal [8].

Definición 1.3. *El* ***problema del vacío*** *consiste en determinar si un lenguaje es vacío.*

Por ejemplo, determinar si el lenguaje formado por los números pares mayores que 5, que son primos es vacío.

En la siguiente sección se definen las gramáticas, un mecanismo que permite generar los elementos de un lenguaje.

1.1.2. Gramáticas

Definición 1.4. *Una* ***gramática*** *es un formalismo utilizado para describir lenguajes formales. Se define como una 4-tupla:*

$$G = (N, \Sigma, P, S),$$

donde:

- N*: Es un conjunto finito de* ***símbolos no terminales****.*
- Σ*: Es un conjunto finito de* ***símbolos terminales****, que constituyen el alfabeto sobre el que se construyen las cadenas del lenguaje. Se cumple que* $N \cap \Sigma = \emptyset$*.*
- P*: Es un conjunto finito de* ***reglas de producción*** *de la forma:*

$$\alpha \to \beta, \quad \textit{donde } \alpha \in (N \cup \Sigma)^* \textit{ y } \beta \in (N \cup \Sigma)^*.$$

- S*: Es el* ***símbolo inicial****,* $S \in N$*, que define el punto de partida para derivar cadenas del lenguaje.*

Una derivación en la gramática consiste en seleccionar una **regla de producción** $\alpha \to \beta$ y sustituir una ocurrencia de α en una cadena w por β [8].

Una cadena $w \in \Sigma^*$ se puede generar por la gramática G si existe una secuencia de derivaciones que comienza con S y termina con la cadena w.

El lenguaje generado por una gramática G se denota como:

$$L(G) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\},$$

donde $\xrightarrow{*}$ indica una derivación en cero o más pasos.

A continuación se presenta la jerarquía de Chomsky, que clasifica a los lenguajes formales de acuerdo con su poder de generación.

1.1.3. Jerarquía de Chomsky

La **Jerarquía de Chomsky** clasifica los lenguajes en cuatro tipos, según las restricciones en sus reglas de producción y la capacidad expresiva de los lenguajes que generan.

El primer tipo de gramáticas son las **Gramáticas irrestrictas**. Estas gramáticas no tienen restricciones en las reglas de producción. Cada regla tiene la forma: $\alpha \to \beta$, donde $\alpha, \beta \in (N \cup \Sigma)^*$ y $\alpha \neq \varepsilon$. Todo lenguaje generado por una gramática irrestricta se denomina **lenguaje recursivamente enumerable**.

El siguiente tipo en la jerarquía de Chomsky son las **Gramáticas dependientes del contexto**. En estas cada regla tiene la forma: $\alpha A \gamma \to \alpha \beta \gamma$, donde $A \in N$, $\alpha, \beta, \gamma \in (N \cup \Sigma)^*$, y $|\beta| \geq 1$. Todo lenguaje generado por una gramática dependiente del contexto se denomina **lenguaje dependiente del contexto**. Todo lenguaje dependiente del contexto es también un lenguaje recursivamente enumerable.

A continuación le siguen las **Gramáticas libres del contexto**. En estas cada regla tiene la forma: $A \to \beta$, donde $A \in N$ y $\beta \in (N \cup \Sigma)^*$. Todo lenguaje generado por una gramática libre del contexto se denomina **lenguaje libre del contexto**. Todo lenguaje libre del contexto es también un lenguaje dependiente del contexto.

El último tipo en la jerarquía son las **Gramáticas regulares**. En estas las reglas de producción tienen la forma:

$$A \to aB \quad \text{o} \quad A \to a,$$

donde $A, B \in N$ y $a \in \Sigma$. Todo lenguaje generado por una gramática regular se denomina **lenguaje regular**. Todo lenguaje regular es un lenguaje libre del contexto.

Un lenguaje A es más expresivo que un lenguaje B si todas las cadenas que pertenecen a B también pertenecen a A y existen cadenas que pertenecen a A que no pertenecen a B, por ejemplo los lenguajes libres del contexto son más expresivos que los lenguajes regulares [8].

Lo anterior se puede ilustrar con el lenguaje *Copy* sobre un alfabeto Σ, que se define como $L_{copy} = \{w^+ \,|\, w \in Z^*\}$. El lenguaje L_{copy} se usa en los capítulos 3 y 4 para construir el lenguaje de las fórmulas booleanas satisfacibles. Si se toma un caso particular de L_{copy}, al cual se le llama $L^n_{copy} = \{w^n \,|\, w \in Z^*\}$, se cumple que L^1_{copy} es un lenguaje regular, mientras $L^k_{copy} \forall k \geq 2$ es un lenguaje dependiente del contexto [8].

En la próxima sección se presentan los autómatas asociados a cada elemento de la jerarquía de Chomsky.

1.2. Autómatas

Un autómata es una máquina abstracta que procesa cadenas de símbolos de un alfabeto finito y determina si una cadena pertenece a un lenguaje [8].

Cada gramática de la jerarquía de Chomsky tiene un autómata equivalente: una Máquina de Turing permite reconocer los lenguajes recursivamente enumerables [8], una Máquina de Turing linealmente acotada permite reconocer los lenguajes dependientes del contexto [8], una Autómata de pila permite reconocer los lenguajes libres del contexto [8] y un Autómata regular permite reconocer los lenguajes regulares [8].

En este trabajo solo se utilizan los autómatas regulares, a partir de los cuales se puede construir un transductor finito, que se usa en el capítulo 3 para sea una asignación de valores a un conjunto de variables binarias generar todas las fórmulas booleanas satisfacibles por dichos valores.

1.2.1. Autómata regular

Definición 1.5. *Un* ***autómata regular*** *[8], también conocido como autómata finito, es un modelo matemático que permite reconocer si una cadena pertenece a un lenguaje regular y se define como una 5-tupla*

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F),$$

donde:

- Q*: Es un conjunto finito de* ***estados****.*
- Σ*: Es el* ***alfabeto*** *finito de entrada.*
- δ*: Es la* ***función de transición****,* $\delta : Q \times \Sigma \to Q$*, que define cómo el autómata cambia de estado en función del símbolo leído.*
- $q_0 \in Q$*: Es el* ***estado inicial*** *desde donde comienza la computación.*
- $F \subseteq Q$*: Es el conjunto de* ***estados de aceptación o estados finales****.*

El autómata comienza en el estado inicial q_0 y procede a leer el primer símbolo de la cadena. En cada paso, la función de transición δ determina, a partir del símbolo actual de la cadena, el siguiente estado al que debe pasar el autómata. Si al finalizar la lectura del último símbolo de la cadena, el autómata se encuentra en un estado de aceptación $q \in F$, entonces la cadena se acepta; en caso contrario, se rechaza.

A continuación se presenta una extensión de los autómatas finitos, que se usa en el capítulo 3 para dada una asignación de valores de las variables, generar todas las fórmulas booleanas satisfacibles por dichos valores.

1.2.2. Transductor finito

Un transductor finito [5] es un modelo computacional que extiende los autómatas finitos al incluir una salida para cada transición.

Definición 1.6. *Un* ***transductor finito*** *es un autómata finito con una función de transición extendida que recibe un símbolo de la cadena de entrada y un estado, y devuelve el estado al cual pasa el transductor y un símbolo. Como resultado de la aceptación de una cadena el transductor genera una cadena de salida formada por todos los símbolos que se obtuvieron como resultado de la función de transición en el proceso de reconocimiento.*

Un transductor finito se define como una 6-tupla:

$$T = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

donde:

- Q *es el conjunto finito de* ***estados****.*
- Σ *es el* ***alfabeto*** *de entrada.*
- Γ *es el* ***alfabeto*** *de salida.*

- $\delta : Q \times \Sigma \rightarrow Q \times \Gamma^*$ *es la **función de transición**, que mapea una combinación de estado actual y símbolo de entrada a un nuevo estado y una salida.*
- $q_0 \in Q$ *es el **estado inicial**.*
- $F \subseteq Q$ *es el **conjunto de estados finales**.*

A modo de ejemplo, se puede definir un transductor finito T, que reconozca cadenas del alfabeto $\{a, b\}$ que tengan ninguna o más a seguida de una o más b y genere una cadena formada por una cantidad de 0 igual a la cantidad de a de la cadena original seguida de una cantidad de 1 igual a la cantidad de b de la cadena original. Ese transductor se describe de la siguiente forma.

$$T = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

donde:

- $Q = \{q_0, q_1, q_2\}$ es el conjunto de estados.
- $\Sigma = \{a, b\}$ es el alfabeto de entrada.
- $\Gamma = \{0, 1\}$ es el alfabeto de salida.
- δ es la función de transición definida por:

δ	a	b
q_0	$(q_0, 0)$	$(q_1, 1)$
q_1	$(q_2, 0)$	$(q_1, 1)$
q_2	$(q_2, 0)$	$(q_2, 1)$

  Las columnas de la tabla representan el símbolo que se lee y las filas el estado en que se encuentra el transductor, los valores de la tabla son una tupla que contiene el estado al cual pasa el transductor y el símbolo que se escribe.

- q_0 es el estado inicial.
- $F = \{q_1\}$ es el conjunto de estados finales.

Para reconocer *aaaabbbbb*, primero T empieza por el estado q_0 y el primer caracter a, entonces pasa al estado q_0 y genera un 0. Luego, al leer el segundo caracter a, se mantiene en el estado q_0 y genera otro 0. Este proceso se repite hasta el quinto caracter que es b, entonces pasa al estado q_1. Luego, al leer el sexto caracter b, se mantiene en el estado q_1 y genera un 1. Este proceso se repite varias veces hasta que se llega al final de la cadena, por tanto se genera la cadena 000011111.

Todas las operaciones con lenguajes y los problemas relacionados con ellos tienen una dificultad y para medir esta dificultad se utiliza un marco teórico llamado complejidad computacional, el cual se presenta en la próxima sección.

1.3. Complejidad computacional

En esta sección se definen los principales conceptos de complejidad computacional: notación asintótica y las clases de problemas. A continuación se presenta una notación para describir el tiempo que demora un algoritmo en realizar determinado cómputo.

1.3.1. Notación asintótica

La notación asintótica se utiliza para describir el comportamiento de una función $f(n)$ a medida que n crece hacia el infinito. Seguidamente se define la notación que será utilizada en el resto del trabajo:

Definición 1.7. *Una función $g(n)$ pertenece a $O(f(n))$ si existen constantes positivas c y n_0 tales que:*

$$g(n) \leq c \cdot f(n) \quad \text{para todo } n \geq n_0.$$

Esta notación proporciona un límite superior asintótico para $g(n)$.

La notación asintótica permite describir el tiempo de ejecución de un algoritmo con respecto al número de operaciones básicas realizadas por un modelo formal de cómputo. Algoritmos como determinar el mínimo y el máximo de un arreglo son $O(n)$, ya que necesitan realizar una cantidad n de operaciones básicas en relación con la cantidad de elementos del arreglo.

Se dice que un algoritmo tiene un tiempo polinomial si puede ejecutarse en una complejidad de $O(n^k)$, donde n es el tamaño de la entrada del algoritmo y k es una constante.

En la próxima sección se presenta una clasificación de los problemas de acuerdo a su complejidad computacional.

1.3.2. Clases de problemas

Los problemas computacionales [8] se agrupan en diferentes clases según los recursos necesarios para resolverlos. En este trabajo se emplean las clases P, NP, NP-completo y NP-duro.

Definición 1.8. *Un problema pertenece a la clase **P** si puede resolverse en tiempo polinomial [8].*

Definición 1.9. *Un problema pertenece a la clase **NP** si su solución puede verificarse en tiempo polinomial [8].*

Definición 1.10. *Un problema pertenece a la clase **NP-Completo**, si pertenece a NP y además es tan difícil como cualquier otro problema en NP. Esto significa que cualquier problema en NP puede reducirse a este problema en tiempo polinómico [8].*

Definición 1.11. *Un problema pertenece a la clase* ***NP-Duro****, si es tan difícil como cualquier otro problema en NP, pero no necesariamente pertenece a NP [8].*

La relación entre las clases P y NP es uno de los problemas abiertos más importantes en la teoría de la computación [8]. Hasta la fecha, se desconoce si P = NP o si P ≠ NP, es decir, no se conoce si realmente los problemas en NP son más difíciles que los problemas en P. Por otro lado, el conjunto de problemas NP-Completo brinda una base sólida para el problema anterior, ya que dada su definición, cualquier problema perteneciente a este conjunto que sea soluble en tiempo polinomial implica que todos los problemas en NP lo son. Mientras que los problemas en NP-Duro pueden resultar aún más difíciles. Es decir aunque resultara que P = NP es posible que existan problemas en NP-Duro que no se puedan resolver en tiempo polinomial [8].

A continuación se presenta el problema que sirve de base a los problemas de la clase NP-Completo: el problema de las satisfacibilidad booleana.

1.4. Problema de la satisfacibilidad booleana

El problema de la satisfacibilidad booleana (SAT), es un problema fundamental en la teoría de la computación y la lógica matemática [8]. El objetivo es determinar si existe una asignación de valores a las variables de una fórmula booleana tal que la expresión sea verdadera.

A continuación se presentan los principales elementos del SAT:

- **Variables booleanas:** Una variable booleana es una variable que puede tomar uno de dos valores posibles: *true* (verdadero) o *false* (falso). Estas variables se utilizan para construir expresiones lógicas.

- **Literales:** Un literal es una variable booleana o su negación. Formalmente, si x es una variable booleana, entonces x y $\neg x$ (la negación de x) son literales. Un literal puede tomar los valores $true$ o $false$ dependiendo de la asignación de valores a las variables.

- **Cláusulas:** Una cláusula es una disyunción (operador **OR**) de uno o más literales. Por ejemplo, la cláusula $(x \vee \neg y \vee z)$ es una disyunción de tres literales: x, $\neg y$ y z. Una cláusula es verdadera si al menos uno de sus literales es verdadero. Si todos los literales son falsos, la cláusula será falsa.

- **Fórmulas en forma normal conjuntiva:** Una fórmula booleana en forma normal conjuntiva (CNF) es una conjunción (operador **AND**) de cláusulas. En otras palabras, es una expresión booleana que se puede escribir como una serie de cláusulas unidas por el operador **AND**. Por ejemplo:

$$(x \vee \neg y \vee z) \wedge (\neg x \vee y) \wedge (x \vee \neg z).$$

- **Fórmulas booleanas equivalentes:** Dos fórmulas booleanas se consideran equivalentes si, para cualquier asignación de valores a sus variables, ambas producen el mismo resultado lógico. Por ejemplo, las fórmulas $x \vee (y \wedge z)$ y $(x \vee y) \wedge (x \vee z)$ son equivalentes, ya que para cualquier combinación de valores x, y, z, ambas tienen el mismo valor lógico.

  Para cualquier fórmula booleana existe una fórmula booleana equivalente en CNF [8] y el algoritmo para encontrarla es polinomial, por lo tanto se puede asumir que toda fórmula booleana está en CNF.

Definición 1.12. *El problema de la* ***satisfacibilidad booleana****, o SAT, consiste en determinar si existe una asignación de valores* $true$ *o* $false$ *a las variables de una fórmula booleana tal que la fórmula completa sea verdadera.*

Una fórmula booleana en CNF es satisfacible si existe una asignación de valores a las variables tal que todas las cláusulas de la fórmula sean verdaderas simultáneamente. Si existe tal asignación, la fórmula es *satisfacible*. Si no existe tal asignación, la fórmula es *insatisfacible*.

1.4.1. SAT como problema NP-Completo y variantes polinomiales

El SAT es el primer problema demostrado como NP-Completo [8] y juega un rol central en la teoría de la complejidad computacional. Se define en la clase NP porque, dada una asignación de valores a las variables de la fórmula booleana, se puede verificar en tiempo polinómico si dicha asignación satisface la fórmula.

Además, la prueba de que SAT es NP-Completo fue una de las contribuciones principales de Stephen Cook en 1971 [8], marcando el inicio de la teoría de la NP-completitud.

Un SAT con exactamente n variables distintas en cada cláusula se denomina n-SAT. Para el problema 2-SAT existe una solución polinomial que determina si la fórmula booleana es satisfacible o no [7], pero para el problema 3-SAT no se conoce ningún algoritmo polinomial que permita determinar si una fórmula booleana es satisfacible o no [8].

Cualquier fórmula booleana del problema n-SAT se puede reducir a una fórmula booleana equivalente del problema 3-SAT, por lo tanto, SAT es equivalente a 3-SAT en términos de complejidad computacional [8].

Aunque no se conoce ningún algoritmo polinomial para resolver el problema SAT en general, existen casos particulares del problema que sí pueden ser resueltos en tiempo polinomial como el 1-SAT, 2-SAT, Horn-SAT, y XOR-SAT.

El problema **1-SAT** es una instancia particular de SAT donde cada cláusula tiene a lo sumo un literal. Este problema puede ser resuelto en tiempo polinomial mediante un algoritmo de asignación de valores de booleanos.

El problema **2-SAT** es una instancia de SAT donde cada cláusula contiene exactamente dos literales. Este problema puede ser resuelto en tiempo polinomial mediante una modelación basada en grafos, utilizando algoritmos como la detección de componentes fuertemente conexas en el grafo de implicación [7].

El problema **Horn-SAT** es una instancia de SAT, donde cada cláusula tiene a lo sumo un literal positivo. Este problema puede ser resuelto en tiempo polinomial mediante el algoritmo de resolución de Horn [6].

El problema **XOR-SAT** es una instancia de SAT donde cada cláusula representa una operación XOR sobre los literales. Puede ser resuelto en tiempo polinomial transformando el problema en un sistema de ecuaciones lineales modulares y aplicando eliminación de Gauss [10].

En la siguiente sección se presentan 2 trabajos que proponen resolver el problema SAT usando elementos de la teoría de lenguajes.

1.5. Solución de instancias del SAT en tiempo polinomial usando teoría de lenguajes

Como parte del estudio del problema SAT, en la Facultad de Matemática y Computación de la Universidad de La Habana se han desarrollado 2 trabajos: [1] y [9], utilizando un enfoque basado en formalismos de la teoría de lenguajes, buscando resolver instancias específicas del SAT, que tienen una solución polinomial.

La idea principal que se aborda en [1] consta de tres partes: asumir que todas las variables en la fórmula son distintas, construir un autómata finito que reconozca cadenas de 0 y 1 que hagan verdadera esa fórmula (asumiendo que todas las variables son distintas), y por último intersectar ese lenguaje regular con un lenguaje libre del contexto que garantice que todas las instancias de la misma variable tenga el mismo valor. Luego de esos tres pasos, se obtiene un lenguaje libre del contexto de las cadenas de 0 y 1 que satisfacen la fórmula y que además respeta los valores de las variables duplicadas.

Para determinar si la fórmula es satisfacible o no, basta con determinar si el lenguaje es vacío, que en el caso de los lenguajes libres del contexto tiene una complejidad $O(n)$. Todo el algoritmo descrito anteriormente tiene una complejidad que es $\mathrm{O}(n^3)$, donde n es el tamaño de la fórmula booleana.

El autómata finito diseñado en [1], se denominó **autómata booleano**. La idea detrás de este es representar las reglas de la lógica proposicional en transiciones entre los estados de un autómata finito, donde cada estado del autómata representa un valor de verdad positivo o negativo que significa que hasta ese momento (solo tomando las instancias de las variables asociadas a los caracteres reconocidos) la fórmula se evalúa positiva o negativa respectivamente [1].

En [9] se generaliza la idea de [1], pero esta vez el autómata booleano asociado a la fórmula booleana se intersecta con una gramática de concatenación de rango simple [4], lo cual permite ampliar el conjunto de fórmulas booleanas que pueden resolverse usando esta estrategia.

En el presente trabajo se sigue otro enfoque para resolver el SAT usando teoría de lenguajes: en vez de resolver el problema del vacío para el lenguaje de todas las interpretaciones que hacen verdadera a una fórmula dada, se construye el lenguaje de todas las fórmulas booleanas satisfacibles y para determinar si un SAT es satisfacible se comprueba si pertenece a ese lenguaje.

Lo anterior permite demostrar que muchos formalismos tienen el problema de la palabra NP-Duro (este conjunto preciso de formalismos se define en el capítulo 3). Por otro lado, se obtiene una gramática de concatenación de rango que reconoce las fórmulas booleanas satisfacibles. A partir de la gramática que reconoce el lenguaje de todas las fórmulas booleanas satisfacibles, se puede demostrar que las gramáticas de concatenación de rango abarcan todos los problemas que pertenecen a la clase NP.

En el siguiente capítulo se presentan las gramáticas de concatenación de rango.

Capítulo 2

Gramáticas de concatenación de rango

Las gramáticas de concatenación de rango (*RCG*) [4] son un formalismo de gramáticas desarrollado en 1988 como una propuesta de Pierre Boullier, un investigador en el campo de la lingüística computacional. Su objetivo principal era proporcionar un modelo más general y expresivo que las gramáticas libres del contexto para describir lenguajes. Las RCG fueron diseñadas con el fin de analizar propiedades y características del lenguaje natural.

Las gramáticas de concatenación de rango se emplean en el capítulo 4 para construir una gramática que reconozca las fórmulas booleanas satisfacibles.

En la próxima sección se presentan algunas nociones que sirven de introducción para las principales definiciones y conceptos de las gramáticas de concatenación de rango.

2.1. Presentación de los elementos de las gramáticas de concatenación de rango

En esta sección se presentan nociones sobre la sustitución de rango y las derivaciones de las RCG, aspectos que sirven de base para los conceptos y definiciones relacionados con las gramáticas de concatenación de rango.

A los no terminales de esta gramática se les llama predicados y cada predicado tiene un conjunto de argumentos. A la cantidad de argumentos de un predicado se le denomina aridad.

Por ejemplo, $A(X,Y)$ representa el no terminal (predicado) A, que tiene como argumentos X e Y. En este caso, la aridad de A es 2.

Cada argumento de los predicados puede estar formado por variables y terminales.

En el caso anterior, X e Y son variables.

Por convenio las variables se denotan por letras mayúsculas del final del alfabeto, y a los terminales, como es usual, con letras minúsculas. Con este convenio, el siguiente predicado: $B(aX,XY,abZ)$, tiene aridad 3. Su primer argumento está formado por el terminal a y la variable X. El segundo argumento por la concatenación de las variables X e Y. El tercer argumento está formado por la concatenación de los terminales a, b y la variable Z.

Cada predicado reconoce un vector de cadenas que tiene como dimensión la aridad del predicado y cada cadena del vector se asocia a un argumento del predicado.

Por ejemplo, si al predicado A se le asocia el vector $[abc,d]$, al primer argumento de A se le asigna la cadena abc y al segundo la cadena d. Por otro lado, si al predicado B se le asigna el vector $[aa,we,abcc]$, al primer argumento de B se le asigna la cadena aa, al segundo we y al tercero $abcc$.

A las producciones de esta gramática se les denomina cláusulas. La parte izquierda de la producción siempre está formada por un único no terminal y en en la parte derecha, pueden existir no terminales con sus respectivos argumentos, o la cadena vacía. Un ejemplo de cláusula puede ser la siguiente:

$$A(XYZ,W) \rightarrow B(X)C(XY,Z)D(W).$$

La regla anterior tiene el siguiente significado: el no terminal A recibe un vector de dimensión 2. A partir de las cadenas del vector, se le asigna valores a las variables X, Y, Z y W, y con esos valores construye los vectores que recibirán los no terminales B, C y D.

El primer argumento de A, es XYZ, lo cual significa todas las formas de dividir la primera cadena del vector que recibe A en 3 subcadenas de la cadena que no se solapen y que su concatenación forme la cadena original. El segundo argumento de A, es W, por tanto se le asigna a W la cadena completa. Por ejemplo, si el no terminal A recibe el vector $[abc,d]$ los valores de X, Y, Z y W pueden ser interpretados de la siguiente manera (Figura 2.1).

Suponga que la interpretación fue la segunda, en la que $X = ab$, $Y = \varepsilon$, $Z = c$, $W = d$, y con esa asignación de variables se evalúa en los no terminales de su parte derecha: $B(X)C(Y,Z)D(W)$, que en este caso sería $B(ab)C(\varepsilon,c)D(d)$. Este proceso se repite en cada uno de los predicados del lado derecho de la cláusula.

Existe otro tipo de producciones que son de la forma $A(X_1,\dots,X_n) \rightarrow \varepsilon$. Por ejemplo, las siguientes cláusulas de la gramática anterior son:

$$B(ab) \rightarrow \varepsilon,$$

$$C(\varepsilon,c) \rightarrow \varepsilon,$$

$$D(d) \rightarrow \varepsilon.$$

X	Y	Z	W
a	b	c	d
ab	ε	c	d
ab	c	ε	d
abc	ε	ε	d
ε	ab	c	d
ε	abc	ε	d
ε	ε	abc	d
a	ε	bc	d
$\vdots$	$\vdots$	$\vdots$	d
a	bc	ε	d

Figura 2.1: Posibles valores de las variables X, Y y Z

Si se continúa el proceso de derivación, $B(ab)$, $C(\varepsilon, c)$ y $D(d)$ derivan en la cadena vacía, cuando esto pasa se dice que B, C y D reconocen los vectores de cadenas $[ab]$, $[\varepsilon, c]$ y $[d]$ respectivamente. A su vez, A reconoce el vector $[abc, d]$ ya que existe una derivación desde $A(abc, d)$ a $B(ab)C(\varepsilon, c)D(d)$ y cada uno de estos predicados deriva en la cadena vacía.

Con la idea anterior se puede hablar del concepto de rango, que no es más que un par de índices i y j, tales que $i \leq j$, estos representan la subcadena de la cadena de entrada que comienza en el i-ésimo caracter y termina en el j-ésimo caracter.

El concepto de rango se utiliza cuando se le asigna una cadena a un argumento de un predicado, lo que significa que a cada variable se le asigna un rango de la cadena, tales que estos no se solapen, como se mostró en el ejemplo anterior.

Dadas estas nociones, a continuación se presentan las principales definiciones de las gramáticas de concatenación de rango.

2.2. Definiciones

En esta sección se define el concepto de rango, gramática de concatenación de rango, sustitución de rango y gramática de concatenación de rango positiva.

Definición 2.1. *Un* ***rango*** *es una tupla (i, j) que representa un intervalo de posiciones en una cadena, donde i y j son enteros no negativos tales que $i \leq j$.*

Por ejemplo, para la cadena *abcd*, el rango $(1, 2)$, representa la subcadena *bc*.

Definición 2.2. *Una* ***gramática de concatenación de rango*** [1] *se define como*

[1] En la literatura este tipo de RCG se toma com gramática de concatenación de rango positiva,

una 5-tupla :

$$G = (N, T, V, P, S),$$

donde:

- N*: Es un conjunto finito de* ***predicados o símbolos no terminales****: Cada predicado tiene una* ***aridad****, que indica la dimensión del vector de cadenas que reconoce y cada cadena del vector se asocia a un argumento del predicado.*

- T*: Es un conjunto finito de* ***símbolos terminales****.*

- V*: Es un conjunto finito de* ***variables****.*

- P*: Es un conjunto finito de* ***cláusulas****, de la forma:*

$$A(x_1, x_2, \ldots, x_k) \to B_1(y_{1,1}, y_{1,2}, \ldots, y_{1,m_1}) \ldots B_n(y_{n,1}, y_{n,2}, \ldots, y_{n,m_n}),$$

  donde $A, B_i \in N$*,* $x_i, y_{i,j} \in (V \cup T)^*$*, y* k *es la aridad de* A*.*

- $S \in N$*: Es el* ***predicado inicial*** *de la gramática, que siempre tiene* ***aridad*** *1.*

Por ejemplo, a continuación se muestra una gramática de concatenación de rango:

$$G^3_{copy} = (N, T, V, P, S),$$

donde:

- N=$\{A, S\}$.
- T=$\{a, b, c\}$.
- V=$\{X, Y, Z\}$.
- El conjunto de cláusulas P es el siguiente:
  1. $S(XYZ) \to A(X, Y, Z)$
  2. $A(aX, aY, aZ) \to A(X, Y, Z)$
  3. $A(bX, bY, bZ) \to A(X, Y, Z)$
  4. $A(cX, cY, cZ) \to A(X, Y, Z)$
  5. $A(\varepsilon, \varepsilon, \varepsilon) \to \varepsilon$
- El símbolo inicial es S.

pero como es la única que se usa en este trabajo se le llama solo gramática de concatenación de rango

Definición 2.3. *Una* ***sustitución de rango*** *es un mecanismo que reemplaza una variable por un rango de la cadena, respetando la estructura del argumento que se asocia a la cadena que se reconoce.*

Por ejemplo, dado el predicado $A(Xa)$ donde $X \in V$ y $a \in T$, la estructura el argumento de A es una variable X seguida del terminal a. Si el no terminal A recibe la cadena baa, X se puede asociar con el rango ba de la cadena original porque si $X = ba$, entonces $Xa = baa$ y esa es justamente la cadena que recibió A.

Por otro lado, la variable X no puede tomar el valor baa, porque ningún caracter de la cadena de entrada coincidiría con el terminal a. De manera similar, X tampoco puede tomar el valor b porque el valor que se asigna a X no cubre la cadena completa.

Definición 2.4. *Las* ***gramáticas de concatenación de rango simple*** *(SRCG) son un subconjunto de las RCG que restringen la forma de las cláusulas de producción. Una RCG G es* ***simple*** *si los argumentos en el lado derecho de una cláusula son variables distintas, y todas estas variables (y no otras) aparecen una sola vez en los argumentos del lado izquierdo.*

Este es un caso particular de las RCG el cual se usa en [9] para describir el orden de las variables de una fórmula booleana.

En la próxima sección se describe el proceso de derivación de las RCG.

2.3. Proceso de derivación

La idea principal para realizar una derivación en la cláusula

$$A(x_1, x_2, \ldots, x_k) \to B_1(y_{1,1}, y_{1,2}, \ldots, y_{1,m_1}) \ldots B_n(y_{n,1}, y_{n,2}, \ldots, y_{n,m_n}),$$

de una RCG, se basa en tomar el vector de cadenas $[w_1, w_2, \ldots, w_k]$ que recibe el predicado A y asociar cada cadena al argumento correspondiente, w_1 se asocia al argumento x_1, w_2 se asocia al argumento x_2 y así hasta que por último w_k se asocia a x_k.

Después se realizan todas las posibles sustituciones en rango para cada argumento y se asocia un rango a cada variable del predicado izquierdo.

Los valores de las variables obtenidos en el paso anterior se instancian en las variables de los predicados del lado derecho de la cláusula.

Por ejemplo, se tiene la cláusula $A(X, aYb) \to B(aXb, Y)$, donde X e Y son variables y a y b son símbolos terminales. Cuando A recibe el vector $[a, abb]$, a coincide con X y abb coincide con aYb. La única sustitución de rango posible es cuando $X = a$ y $Y = b$, por tanto el predicado $A(a, abb)$ deriva como $B(aab, b)$.

Las RCG, a diferencia de las gramáticas definidas en la sección 1.1.2 del capítulo 1 no generan cadenas, su funcionamiento se basa en reconocer si una cadena pertenece o no al lenguaje.

Un vector de cadenas se reconoce por un predicado A si existe una secuencia de derivaciones que comienza en A y termina en la cadena vacía.

Por ejemplo, dada la cláusula $A(X_1, X_2, X_3) \to B_1(X_1)B_2(X_2)B_3(X_3)$, el vector $[w_1, w_2, w_3]$ se reconoce por A, si existe una secuencia de derivaciones para cada uno de los predicados $B_1(w_1)$, $B_2(w_2)$, $B_3(w_3)$ que derive en la cadena vacía.

A continuación se presenta un ejemplo de reconocimiento de la cadena *abcabcabc* por la gramática G^3_{copy} presentada en la página 18.

La cadena *abcabcabc* se reconoce por G^3_{copy}, ya que $S(abcabcabc)$ se puede derivar de la siguiente manera:

$$S(abcabcabc) \to A(abc, abc, abc) \to A(bc, bc, bc) \to A(c, c, c) \to A(\varepsilon, \varepsilon, \varepsilon) \to \varepsilon.$$

El proceso de reconocimiento de la cadena *abcabcabc* por la gramática G^3_{copy} se detalla como sigue.

- **Primer paso:** En la primera cláusula $S(XYZ) \to A(X, Y, Z)$, la sustitución de rango asocia las variables X, Y, Z a los valores $X = abc$, $Y = abc$ y $Z = abc$ y se deriva en el predicado $A(abc, abc, abc)$.

- **Segundo paso:** En la segunda cláusula $A(aX, aY, aZ) \to A(X, Y, Z)$, la sustitución de rango asocia las variables X, Y, Z a los valores $X = bc$, $Y = bc$ y $Z = bc$ y se deriva en el predicado $A(bc, bc, bc)$.

- **Tercer paso:** En la tercera cláusula $A(bX, bY, bZ) \to A(X, Y, Z)$, la sustitución de rango asocia las variables X, Y, Z a los valores $X = c$, $Y = c$ y $Z = c$ y se deriva en el predicado $A(c, c, c)$.

- **Cuarto paso:** En la cuarta cláusula $A(cX, cY, cZ) \to A(\varepsilon, \varepsilon, \varepsilon)$, la sustitución de rango asocia las variables X, Y, Z a los valores $X = \varepsilon$, $Y = \varepsilon$ y $Z = \varepsilon$ y se deriva en el predicado $A(\varepsilon, \varepsilon, \varepsilon)$.

- **Quinto paso:** Finalmente en el último paso se toma la última cláusula $A(\varepsilon, \varepsilon, \varepsilon) \to \varepsilon$ que deriva en la cadena vacía, por lo que de esta manera se reconoce la cadena *abcabcabc*.

Se puede demostrar que la gramática G^3_{copy}, reconoce el lenguaje $L^3_{copy} = \{www \mid w \in \{a, b, c\}^*\}$.

A continuación se presentan las propiedades de las RCG que demuestran que las RCG no son cerradas bajo transducción finita, esto no permite construir el lenguaje de todas las fórmulas satisfacibles mediante una transducción finita usando una RCG, lo anterior se analiza en el capítulo 4.

2.4. Propiedades de las RCG

En esta sección se describen las principales propiedades de las RCG que demuestran que las RCG no son cerradas bajo transducción finita. Además en esta sección se presenta el problema de la palabra para las RCG, el cual se emplea en el capítulo 4 para determinar si una fórmula booleana es satisfacible.

- **No cerradas bajo homomorfismo:** Dada una RCG G, el homomorfismo de un lenguaje que se reconoce por G necesariamente no se reconoce por una RCG [2].
- **No cerradas bajo transducción finita:** Dada una RCG G, la transducción finita de un lenguaje que se reconoce por G necesariamente no se reconoce por una RCG. Esto es una consecuencia de la propiedad anterior porque un homomorfismo es un transductor finito de un solo estado y tantas transiciones hacia el mismo estado como transformaciones de símbolos en el homomorfismo.

En este trabajo se propone una forma para construir el lenguaje de todos los SAT satisfacibles, y esto depende de un formalismo que sea capaz de describir una variante de L_{copy} y sea cerrado bajo transducción finita. Sin embargo, esta forma de construir el lenguaje es solo suficiente y no necesaria porque existen formalismos que no son cerrados bajo transducción finita (como las RCG) que también describen el lenguaje.

En [4] se menciona que en la mayoría de los casos el problema de la palabra para las RCG es polinomial y se resuelve mediante un algoritmo de memorización sobre las cadenas asignadas a los argumentos de los predicados de la RCG [4]. Como la cantidad máxima de rangos de la cadena es n^2 y la máxima aridad de un predicado es constante, este proceso de memorización cuenta con una cantidad polinomial de estados, y tiene una complejidad de $O(|P|n^{2h(l+1)})$ donde h es la máxima aridad en un predicado, l es la máxima cantidad de predicados en el lado derecho de una cláusula y n es la longitud de la cadena que se reconoce.

Sin embargo, existen casos en los que el problema de la palabra no es polinomial, en la siguiente sección se analiza un caso en el que este problema no es polinomial.

2.4.1. Problema de la palabra no polinomial para las RCG

El algoritmo de reconocimiento que se menciona en la sección anterior utiliza un proceso de memorización sobre los rangos de la cadena. La idea fundamental para esto y lo que acota la complejidad del algoritmo es que la cantidad de estados asociados a la memorización es igual a la cantidad de rangos de la cadena, el cual es polinomial con respecto a la longitud de la cadena. Esto se cumple siempre que todos los argumentos que reciben todos los no terminales de la gramática sean subcadenas de la cadena original que se está analizando. Existen gramáticas de concatenación de rango en que esto no ocurre, como en la que se muestra a continuación.

La siguiente RCG reconoce el lenguaje $L = \{w \,|\, w \in \{0,1\}^*\}$, esta no tiene uso real porque existe otra RCG equivalente que reconoce el mismo lenguaje, pero ilustra una RCG donde se generan cadenas que no son subcadenas de la cadena de entrada durante el proceso de reconocimiento:

$$G = (N,T,V,P,S),$$

donde:

- N=$\{A,B,Eq,S\}$.
- T=$\{0,1\}$.
- V=$\{X,Y\}$.
- El conjunto de cláusulas P es el siguiente:
  1. $S(X) \to A(X,X)$
  2. $A(1X,Y) \to B(X,0,Y)$
  3. $A(1X,Y) \to B(X,1,Y)$
  4. $A(0X,Y) \to B(X,1,Y)$
  5. $A(0X,Y) \to B(X,0,Y)$
  6. $B(1X,Y,Z) \to B(X,1Y,Z)$
  7. $B(1X,Y,Z) \to B(X,0Y,Z)$
  8. $B(0X,Y,Z) \to B(X,0Y,Z)$
  9. $B(0X,Y,Z) \to B(X,1Y,Z)$
  10. $B(\varepsilon,Y,Z) \to Eq(Y,Z)$
- El símbolo inicial es S.

Para procesar una cadena w, la gramática anterior genera todas las posibles cadenas q, tales que $|w| = |q|$ y luego comprueba si $w = q$.

Esta gramática no tiene caso de uso ya que para toda cadena w siempre va a existir una cadena q tal que $w = q$, por lo que se puede modelar con solamente la cláusula $S(X) \to \varepsilon$. Pero la complejidad del reconocimiento de G es mayor que 2^n (con n igual al tamaño de la cadena de entrada), ya que esta es la cantidad de cadenas posibles que puede recibir el segundo argumento del predicado B, porque la gramática es ambigua y en cada derivación de B existen 2 posibles decisiones, se añade un 1 delante al valor de la Y o se añade un 0. Las siguientes cláusulas ilustran lo planteado anteriormente:

$$\begin{aligned} B(1X, Y, Z) &\to B(X, 1Y, Z), \\ B(1X, Y, Z) &\to B(X, 0Y, Z), \\ B(0X, Y, Z) &\to B(X, 0Y, Z), \\ B(0X, Y, Z) &\to B(X, 1Y, Z). \end{aligned}$$

En el capítulo 4 se presenta una RCG que reconoce fórmulas booleanas satisfacibles, pero es ambigua, con el problema de la palabra no polinomial.

En este capítulo se analizaron las principales definiciones y propiedades de las RCG, que son utilizadas en el capítulo **??** para definir una gramática que reconozca las fórmulas booleanas satisfacibles. En el próximo capítulo se presenta un primer enfoque para definir el lenguaje de todas las fórmulas booleanas satisfacibles y a esta idea se le da continuidad en el capítulo 4, mediante las RCG.

Capítulo 3

Lenguaje de las fórmulas booleanas satisfacibles empleando transducción finita

En este capítulo se presenta el lenguaje L_{S-SAT}, al cual pertenecen todos los problemas SAT que son satisfacibles, y se muestra una forma de construirlo a partir de una transducción finita de una variante del lenguaje L_{copy} sobre el alfabeto $\{0, 1, d\}$. Este lenguaje permitiría resolver instancias del SAT resolviendo el problema de la palabra.

Para definir el lenguaje L_{S-SAT} se presenta una vía para codificar una fórmula booleana mediante cadenas sobre el alfabeto $\Sigma = \{a, b, c, d\}$, y para construirlo se utiliza una transducción finita del lenguaje $L_{0,1,d}$, que es una variante del lenguaje L_{copy} sobre el alfabeto $\{0, 1, d\}$.

La estructura de este capítulo es la siguiente: en la sección 3.1 se muestra como codificar una fórmula booleana cualquiera usando el alfabeto $\{a, b, c, d\}$. En la sección 3.3 se muestra cómo interpretar las cadenas sobre el alfabeto $\{0, 1, d\}$ como asignaciones de las variables. Finalmente, en la sección 3.4.1 se presenta un transductor finito que convierte cadenas del lenguaje $L_{0,1,d}$ en cadenas sobre el alfabeto $\{a, b, c, d\}$ que representan fórmulas booleanas satisfacibles. Seguidamente se conjetura por qué la representación del lenguaje de las fórmulas booleanas satisfacibles, en cualquier formalismo que lo genere usando la estrategia propuesta en este capítulo, tiene un tamaño $O(1)$. Esto implica que el problema de la palabra para todos estos formalismos es NP-Duro.

A continuación se presenta cómo codificar una fórmula booleana cualquiera mediante una cadena sobre el alfabeto $\{a, b, c, d\}$.

3.1. Codificación de una fórmula booleana a una cadena

Una fórmula booleana F, con v variables en CNF tiene la siguiente estructura:

$$F = X_1 \wedge X_2 \wedge \ldots \wedge X_n$$

donde cada cláusula X_i es una disyunción de literales

$$X_i = L_{i1} \vee L_{i2} \vee \ldots \vee L_{im},$$

cada literal L_{ij} es una variable booleana o su negación. También se asume que $m \leq v$.

Si se tiene una fórmula booleana F en forma normal conjuntiva se puede considerar que cada una de las v variables aparece en cada cláusula en uno de tres posibles estados: sin negar, aparece negada, o no aparece.

Por ejemplo, en la primera cláusula de la siguiente fórmula booleana en CNF con 3 variables:

$$F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

la variable x_1 aparece sin negar, la variable x_2 aparece negada, y la variable x_3 no aparece.

El hecho de que se pueda asumir que en todas las cláusulas aparecen todas variables permite representar una cláusula de una fórmula con v variables como una cadena de v símbolos, donde el símbolo en la posición i indica el estado de la variable x_i en la cláusula.

En este trabajo se propone usar los símbolos a, b y c para indicar el estado de una variable en una cláusula, usando el siguiente convenio:

- a: indica que la variable aparece sin negar.
- b: indica que la variable aparece negada.
- c: indica que la variable no aparece.

Con este convenio, la primera cláusula de F se puede representar mediante la cadena abc.

Una vez que se tiene cómo representar una cláusula es posible representar varias cláusulas usando otro símbolo como separador. Por ejemplo se puede usar d, este símbolo indica el final de una cláusula, una fórmula lógica con v variables y k cláusulas se puede representar mediante k bloques de longitud v, donde cada bloque está formado por los símbolos a, b, o c, y cada bloque se separa del siguiente por el símbolo d.

Con este convenio, la fórmula

$$F = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

se representa mediante la cadena:

$$abc\mathbf{d}baa\mathbf{d}aba\mathbf{d},$$

donde los símbolos **d** aparecen en negrita para facilitar la interpretación de la cadena como fórmula en forma normal conjuntiva.

Para que una cadena w pueda ser interpretada como una fórmula booleana debe cumplir con las siguientes condiciones: tener n bloques separados por d, cada bloque de la misma longitud v y en cada bloque solo pueden se pueden usar los caracteres a, b y c. Si cumple con estas características, una cadena w se interpreta como una fórmula booleana con n cláusulas y v variables, donde la estructura de cada cláusula depende de los caracteres correspondiente al bloque de a, b y c que se asocia a dicha cláusula.

Por ejemplo la cadena $w = acc\mathbf{d}aba\mathbf{d}cba\mathbf{d}$, tiene 3 bloques separados por d, los cuales son acc, aba y cba, los 3 tienen tamaño 3 y solo tienen los caracteres a, b y c. Por tanto w se puede interpretar como la siguiente fórmula booleana:

$$(x_1) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$$

Definición 3.1. *Una vez definida la transformación de una fórmula booleana en una cadena, se puede definir el* ***lenguaje de todas las fórmulas booleanas en CNF*** *como:*

$$L_{FULL-SAT} = \{q_1 d q_2 d \dots q_n d \mid q_i \in \{a,b,c\}^+,\ |q_i| = |q_j| \forall i,j = 1 \dots n,\ y\ n \in \mathbb{N}\}.$$

En la próxima sección se define el lenguaje L_{S-SAT} que contiene todas las fórmulas booleanas satisfacibles.

3.2. Definición de L_{S-SAT}

Definición 3.2. *El lenguaje de todas las fórmulas booleanas en CNF que son satisfacibles se define como todas las cadenas $e \in L_{FULL-SAT}$, tales que la fórmula booleana que representa e, sea satisfacible.*

En la próxima sección se demuestra que L_{S-SAT} no es un lenguaje libre del contexto, por lo que el formalismo que lo genere necesariamente debe pertenecer a las gramáticas dependientes del contexto o a las gramáticas irrestrictas.

3.2.1. Demostración de que L_{S-SAT} no es un lenguaje libre del contexto

En esta sección se demuestra que el lenguaje L_{S-SAT} no es libre del contexto. Una técnica que se emplea para demostrar que un lenguaje no es libre del contexto, es el lema del bombeo para los lenguajes libres del contexto [8].

El lema del bombeo establece que si L es un lenguaje libre del contexto existe una constante n tal que para toda cadena $t \in L$, donde $|t| \geq n$, puede escribirse de la forma $t = uvwxy$ tal que: $|vwx| \leq n$, $vx \neq \varepsilon$ y $\forall i \geq 0\, uv^iwx^iy \in L$ [8].

Demostración de que L_{S-SAT} no es un lenguaje libre del contexto. Suponga que L_{S-SAT} es un lenguaje libre del contexto y sea un homomorfismo $h : \{a,b,c\} \to \{1,d\}^*$, tal que $h(a) = 1$, $h(b) = 1$, $h(c) = 1$ y $h(d) = d$. Se define el lenguaje $L_h = \{h(e) \mid e \in L_{S-SAT}\}$, se cumple que L_h es un lenguaje libre del contexto, porque los lenguajes libres del contexto son cerrados bajo homomorfismo [8].

Si una cadena $t \in L_h$, se cumple que t está formada por una cadena de unos concatenada con una d repetida varias veces. Ya que la cadena $e \in L_{S-SAT}$, tal que $h(e) = t$, esta formada por cadenas sobre el alfabeto $\{a,b,c\}$ de la misma longitud concatenadas con una d intermedia.

Sea n la constante asociada a L_h, entonces se cumple que la fórmula

$$(x_1) \wedge (x_2) \wedge \ldots \wedge (x_{n+2})$$

es satisfacible y tiene $n+2$ variables, por lo que la cadena

$$e = a\underbrace{c\ldots c}_{n+1}\mathbf{d}ca\underbrace{c\ldots c}_{n}\mathbf{d}\ldots\underbrace{c\ldots c}_{n}ac\mathbf{d}\underbrace{c\ldots c}_{n+1}a\mathbf{d},$$

pertenece a L_{S-SAT}, entonces la cadena

$$h(e) = \underbrace{11\ldots 1}_{n+2}\mathbf{d}\underbrace{11\ldots 1}_{n+2}\mathbf{d}\ldots\underbrace{11\ldots 1}_{n+2}\mathbf{d}\underbrace{11\ldots 1}_{n+2}\mathbf{d},$$

pertenece a L_h y cada bloque de unos de $h(e)$ tiene $n+2$ caracteres.

Por tanto, como $|h(e)| \geq n$ existen u, v, w, x y y con $vx \neq \varepsilon$ y $|vwx| \leq n$ tales que $h(e) = uvwxy$.

Como $|vwx| \leq n$ y cada bloque de unos tiene $n+2$ caracteres se cumple que vwx tiene a lo sumo una d, porque entre dos caracteres d hay como mínimo $n+2$ caracteres. Entonces pueden darse 2 casos: v o x contienen una d o, ninguno de las 2 contiene una d.

En el primer caso cuando se bombea v y x en la cadena uv^2wx^2y se agrega un bloque de unos más y el bloque de unos que se agrega tiene un tamaño menor o igual a n, por lo que $uv^2wx^2y \notin L_h$, ya que hay un bloque de unos con menos caracteres que las demás.

En el segundo caso cuando se bombea v y x en la cadena uv^2wx^2y se agrega al menos un caracter al bloque de unos al que pertenecía v o al menos un caracter al bloque de unos al que pertenecía x. Si v y x pertenecían al mismo bloque de unos en $uvwxy$ entonces hay un bloque de unos en uv^2wx^2y que tiene más caracteres que los restantes bloques, en caso contrario hay uno o dos bloques en uv^2wx^2y que tienen más caracteres que los restantes bloques, entonces $uv^2wx^2y \notin L_{S-SAT}$.

En los dos casos se cumple que $uv^2wx^2y \notin L_h$ por lo tanto se cumple que L_h no es un lenguaje libre del contexto, luego el lenguaje L_{S-SAT} tampoco es libre del contexto. □

Seguidamente, se muestra cómo interpretar determinados tipos de cadenas de 0 y 1 como la asignación de los valores de las variables de una fórmula booleana.

3.3. El lenguaje $L_{0,1,d}$ como asignación de los valores de las variables de una fórmula booleana

En esta sección se muestra cómo interpretar una cadena w como la asignación de valores para las variables de una fórmula booleana.

Definición 3.3. *El lenguaje $L_{0,1,d}$ se define como:*

$$L_{0,1,d} = \{(wd)^+ \,|\, w \in \{0,1\}^+\}.$$

$L_{0,1,d}$ contiene cadenas sobre el alfabeto $\{0,1,d\}$, que representan una cadena binaria concatenada con una d, repetida varias veces.

Para cada cadena binaria se le asocia el valor del i-ésimo caracter al valor de la i-ésima variable de la cláusula, si este caracter es un 1 se le asocia un valor de *true* (verdadero) y si el caracter es un 0 se le asocia el valor de *false* (falso). Como todas las cadenas binarias son iguales se garantiza que a dos instancias de la misma variable en dos cláusulas distintas se les asigne el mismo valor.

Si se tiene una cadena $e \in L_{FULL-SAT}$, la cual representa una fórmula booleana F en CNF con v variables, todas las cadenas $(wd)^n$, tales que $|w| = v$ y $|(wd)^n| = |e|$, representan todas las posibles interpretaciones de F.

Por ejemplo, si $r = 101\mathbf{d}101\mathbf{d}101\mathbf{d}$ y $e = abc\mathbf{d}cbb\mathbf{d}acc\mathbf{d}$, fórmula booleana:

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

se asignan los valores a las variables de de la siguiente manera: $x_1 = true$, $x_2 = false$ y $x_3 = true$, y la fórmula booleana se evalúa con valor *true*.

Seguidamente se muestra como construir L_{S-SAT} mediante una transducción finita del lenguaje $L_{0,1,d}$.

3.4. Construcción del L_{S-SAT} usando transducción finita

La idea para construir L_{S-SAT} es obtener un transductor finito, denominado T_{SAT}, que acepte como entrada cadenas $r \in L_{0,1,d}$ y devuelva cadenas de $e \in L_{FULL-SAT}$ tales que al evaluar r en e (como se describió en la sección 3.3), e es verdadera.

Se construye L_{S-SAT} como el lenguaje de todas las transducciones e que se obtienen del transductor T_{SAT}, a partir del lenguaje de cadenas de entrada $L_{0,1,d}$.

$$L_{S-SAT} = \{e \,|\, \exists r \in L_{0,1,d} \text{ y } e \in T_{SAT}(r)\}.$$

A continuación se define el transductor T_{SAT}.

3.4.1. Transductor T_{SAT}

En esta sección se define el transductor finito T_{SAT} (Figura 3.2), el cual se usa para construir L_{S-SAT}, mediante una transducción finita del lenguaje $L_{0,1,d}$.

Para definir T_{SAT}, se construye el transductor T_{CLAUSE} (Figura 3.1) que dada una cadena binaria w, genera todas las posibles cláusulas satisfacibles por los valores de las variables que determina la cadena de entrada.

Este transductor tiene 3 estados: el estado inicial, el estado positivo representa que la cláusula generada ya se evalúa con un valor de verdad positivo y el estado negativo representa que la cláusula generada aún no se evalúa con un valor de verdad positivo para los caracteres que se leyeron hasta el momento. Las transiciones entre los estados se realizan dependiendo de si la asignación que se realiza en el momento de leer y de escribir satisface la cláusula generada o no.

A continuación de describe el funcionamiento de cada estado del transductor T_{CLAUSE}:

- Estado q_0: representa el estado inicial. Si se lee un 1 y se escribe una a, se pasa al estado positivo, ya que se genera una variable sin negar a la cual se le asigna el valor *true*. Si se lee un 1 y se escribe una b, se pasa al estado negativo, ya que se genera una variable negada a la cual se le asigna el valor *true*. Si se lee un 1 y se escribe una c, se mantiene en el mismo estado, ya que se genera una variable que no está en la cláusula a la cual se le asigna el valor *true*. Si se lee un 0 y se escribe una a, se pasa al estado negativo, ya que se genera una variable sin negar a la cual se le asigna el valor *false*. Si se lee un 0 y se escribe una b, se pasa al estado positivo, ya que se genera una variable negada a la cual se le asigna el valor *false*. Si se lee un 0 y se escribe una c, se mantiene en el mismo estado, ya que se genera una variable que no está en la cláusula a la cual se le asigna el valor *false*.

- Estado q_p (estado positivo de T_{CLAUSE}): representa que para los valores asignados a las variables se obtiene un valor de verdad positivo. Como la fórmula se encuentra ya en un estado positivo lo que significa que al menos un literal se evaluó positivo, no importa la entrada ni lo que el transductor escriba, se mantiene en el mismo estado. Este estado es el estado de aceptación para el transductor lo cual significa que la cláusula se evalúa con un valor de verdad positivo.

- Estado q_n (estado negativo de T_{CLAUSE}): representa que para los valores asignados a las variables hasta el momento se obtiene un valor de verdad negativo. Si se lee un 1 y se escribe una a, se pasa al estado positivo, ya que se genera una variable sin negar a la cual se le asigna el valor *true*. Si se lee un 1 y se escribe una b, se mantiene en el mismo estado, ya que se genera una variable negada a la cual se le asigna el valor *true*. Si se lee un 1 y se escribe una c, se mantiene en el mismo estado, ya que se genera una variable que no está en la cláusula a la cual se le asigna el valor *true*. Si se lee un 0 y se escribe una a, se mantiene en el mismo estado, ya que se genera una variable sin negar a la cual se le asigna el valor *false*. Si se lee un 0 y se escribe una b, se pasa al estado positivo, ya que se genera una variable negada a la cual se le asigna el valor *false*. Si se lee un 0 y se escribe una c, se mantiene en el mismo estado, ya que se genera una variable que no está en la cláusula a la cual se le asigna el valor *false*.

Seguidamente se define T_{CLAUSE}.

$$T_{CLAUSE} = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

donde:

- $Q = q_0, q_p, q_n$.
- $\Sigma = 0, 1$.
- $\Gamma = a, b, c$.
- $\delta : Q \times \Sigma \to Q \times \Gamma^*$ función de transición.
- $q_0 = q_0$ estado inicial.
- $F = q_p$ conjunto de estados finales.

Se define la función de transición δ de la siguiente manera:
Las transiciones para el estado q_0 son las siguientes:

- $\delta_{SAT}(q_0, 1) = (q_p, a)$
- $\delta_{SAT}(q_0, 0) = (q_n, a)$
- $\delta_{SAT}(q_0, 1) = (q_n, b)$
- $\delta_{SAT}(q_0, 0) = (q_p, b)$
- $\delta_{SAT}(q_0, 1) = (q_0, c)$
- $\delta_{SAT}(q_0, 0) = (q_0, c)$

Las transiciones para el estado q_p (estado positivo de T_{CLAUSE}) son las siguientes:

- $\delta_{SAT}(q_p, 1) = (q_p, a)$
- $\delta_{SAT}(q_p, 0) = (q_p, a)$
- $\delta_{SAT}(q_p, 1) = (q_p, b)$
- $\delta_{SAT}(q_p, 0) = (q_p, b)$
- $\delta_{SAT}(q_p, 1) = (q_p, c)$
- $\delta_{SAT}(q_p, 0) = (q_p, c)$

Las transiciones para el estado q_n (estado negativo de T_{CLAUSE}) son las siguientes:

- $\delta_{SAT}(q_n, 1) = (q_p, a)$
- $\delta_{SAT}(q_n, 0) = (q_n, a)$
- $\delta_{SAT}(q_n, 1) = (q_n, b)$
- $\delta_{SAT}(q_n, 0) = (q_p, b)$
- $\delta_{SAT}(q_n, 1) = (q_n, c)$
- $\delta_{SAT}(q_n, 0) = (q_n, c)$

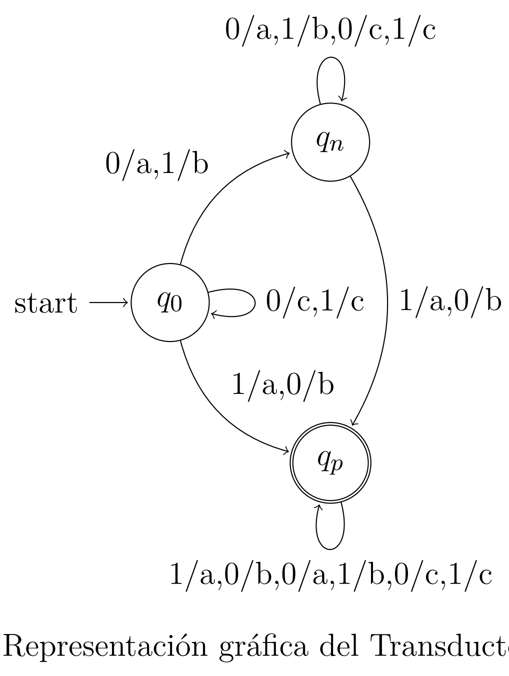

Figura 3.1: Representación gráfica del Transductor T_{CLAUSE}.

Se puede construir T_{SAT} mediante una modificación de T_{CALUSE}. Esta funciona de la siguiente manera: cuando se genera una cláusula y el transductor termina en el estado positivo se continúa leyendo los valores de la cadena de entrada, ya que la cadena de entrada está conformada por varias cláusulas separadas mediante el caracter d que son necesarios para generar la próxima cláusula.

La modificación de T_{CALUSE} se logra definiendo q_0 como el estado de aceptación y agregando una transición del estado q_p al estado q_0 que lea una d y escriba una d. De esta manera, cuando una cláusula se genera con un valor de verdad positivo se comienza a generar la siguiente cláusula desde el estado inicial.

Esta modificación de T_{CALUSE} tiene un inconveniente y es que el transductor genera la cadena vacía, y la cadena vacía representa una fórmula booleana con 0 variables, por lo que no tiene sentido que se considere en L_{S-SAT}. Para solucionar esto se pueden concatenar 2 transductores T_{CLAUSE} y unirlos mediante una transición, esta idea se expone a continuación.

Para definir el transductor T_{SAT} (Figura 3.2) se concatenan 2 transductores T_{CLAUSE} (T_1 y T_2 respectivamente). Sean los estados: q_{0_1}, q_{p_1} y q_{n_1} estado inicial, positivo y negativo de T_1, respectivamente, y los estados q_{0_2}, q_{p_2} y q_{n_2} estado inicial, positivo y negativo de T_2, respectivamente. T_1 y T_2 se concatenan añadiendo una transición de q_{p_1} a q_{0_2} con el símbolo d (tanto de lectura como de escritura) y además se agrega una transición de q_{p_2} a q_{0_2} con el símbolo d (tanto de lectura como de escritura). Para terminar se definen el estado inicial y el estado final de T_{SAT}, los cuales serían q_{0_1} y q_{0_2}, respectivamente.

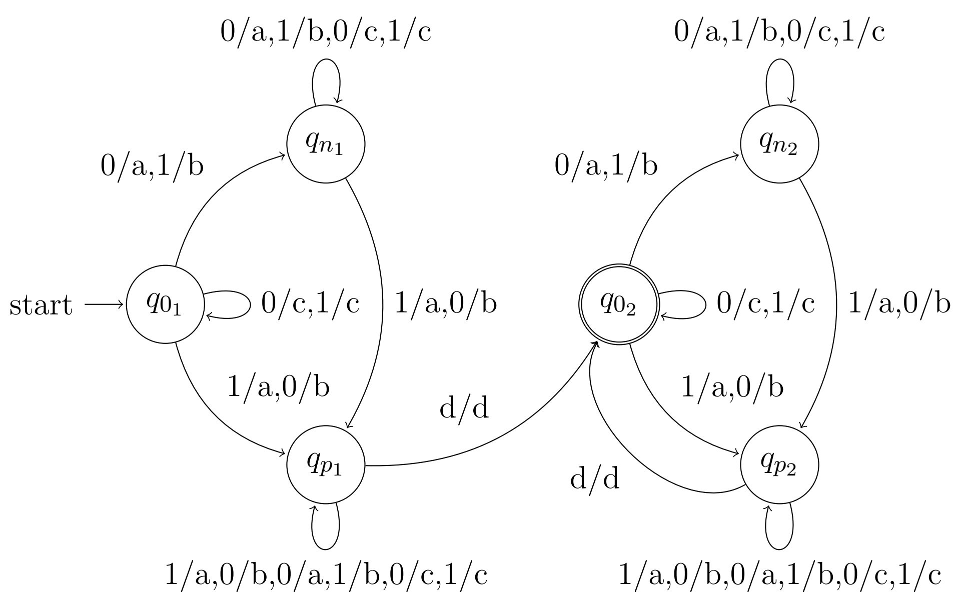

Figura 3.2: Representación gráfica del Transductor T_{SAT}.

A continuación se presenta la demostración de que la construcción de L_{S-SAT} mediante una transducción finita genera todas las fórmulas booleanas satisfacibles.

3.5. Demostración de que la construcción de L_{S-SAT} mediante una transducción finita genera todas las fórmulas satisfacibles

En esta sección se demuestra que la construcción de L_{S-SAT}, mediante el transductor T_{SAT}, genera todas las fórmulas booleanas satisfacibles.

Teorema 3.1. *Una cadena e pertenece al lenguaje generado por la transducción finita del lenguaje $L_{0,1,d}$ mediante el transductor T_{SAT}, si y solo si la fórmula booleana asociada a la cadena e es satisfacible. Esto significa que:*

$$L_{S-SAT} = \{e \,|\, \exists r \in L_{0,1,d} \text{ y } e \in T_{SAT}(r)\}.$$

A continuación, se presentan algunas definiciones que serán usadas en la demostración del Teorema 3.1.

Definición 3.4. *Sea una cadena $w \in \{0,1\}^*$, los valores de w se le asignan a una fórmula booleana F, cuando w tiene la misma longitud que el número de variables de F. Además si el i-ésimo caracter de w es un 1 a la i-ésima variable de F se le asigna el valor true y si el i-ésimo caracter de w es un 0 a la i-ésima variable de F se le asigna el valor false.*

Definición 3.5. *Una cadena binaria w satisface una fórmula booleana F si al asignarle los valores de w a F, se obtiene un valor de verdad positivo.*

Definición 3.6. *Una cadena binaria w satisface a una cadena $e \in L_{FULL-SAT}$ si w satisface la fórmula booleana asociada a e.*

Para la demostración del teorema 3.1 se presentan los siguientes lemas:

Lema 3.1. *Dada una cadena binaria w, $T_{CLAUSE}(w)$ es el conjunto de todas las cadenas que representan cláusulas que son satisfacibles por w.*

Lema 3.2. *Dada una cadena $r = (wd)^n$, con $w \in \{0,1\}^*$, $T_{SAT}(r)$ contiene todas las cadenas que representan fórmulas de n cláusulas satisfacibles por la cadena w.*

La idea para la demostración del Teorema 3.1, es probar que dada una cadena binaria w, $T_{CLAUSE}(w)$ es el conjunto de todas las cadenas que representan cláusulas que son satisfacibles por w, Lema 3.1. Después, dada una cadena $r = (wd)^n$ se realiza una inducción sobre n, para demostrar que $T_{SAT}(r)$ contiene todas las cadenas que representan fórmulas de n cláusulas satisfacibles por la cadena w, Lema 3.1. Para finalizar la demostración se prueba que la transducción de $L_{0,1,d}$ mediante T_{SAT}, genera todas las cadenas $e \in L_{FULL-SAT}$, tales que e representa una fórmula booleana satisfacible, Teorema 3.1.

Seguidamente, se demuestra el Lema 3.1.

Demostración del Lema 3.1. Para demostrar que dada una cadena binaria w, $T_{CLAUSE}(w)$ es el conjunto de todas las cláusulas que son satisfacibles por w, primero suponga que $q \in T_{CLAUSE}(w)$. Esto significa que el transductor terminó en el estado q_p en el proceso que generó q y como empezó en el estado q_0 ocurrió una transición desde q_0 a q_p o desde q_n a q_p.

Esto solo es posible si el transductor leyó un 1 y escribió una a o si leyó un 0 y escribió una b. Entonces, cuando se le asignan los valores de w a las variables de la fórmula booleana que representa q hay una variable sin negar con valor *true* o una variable negada con valor *false*, por lo tanto, se cumple que w satisface la cláusula que representa q.

Sea una cláusula F satisfacible por una cadena binaria w, cuya representación en $L_{FULL-SAT}$ es q, entonces se cumple que cuando se le asignan los valores de w a las variables de F, hay una variable sin negar con valor *true* o una variable negada con valor *false*. Sin pérdida de la generalidad se asume que la primera variable que cumple lo anterior es la i-ésima.

Como en cada estado para cada símbolo que se lee existe una transición que escribe una a, una b o una c; si se hace el reconocimiento de los primeros $i-1$ caracteres de la cadena de entrada por T_{CLAUSE} se pueden seguir las transiciones entre los estados del transductor de tal manera que los primeros $i-1$ caracteres de la cadena generada sean iguales a los primeros $i-1$ caracteres de q.

Dado este punto solo es posible que el transductor esté en el estado q_0 o q_n, pero como se cumple que la i-ésima variable está sin negar y con valor *true* o está negada y con valor *false*, entonces se puede tomar la opción de leer un 1 y escribir una a o leer un 0 y escribir una b según corresponda. De esta manera, según en el estado en que se encuentre el autómata, se pasa al estado q_p.

Luego, se toman las restantes transiciones de manera que la cadena generada sea q y se mantiene en el mismo estado ya que q_p solo tiene transiciones hacia sí mismo. De esta manera se demuestra el Lema 3.1.

□

Seguidamente se demuestra el Lema 3.2.

Demostración del Lema 3.2. Para demostrar que $T_{SAT}(r)$ contiene todas las fórmulas satisfacibles por la cadena $r = (wd)^n$ donde $w \in \{0,1\}^*$ se hará una inducción sobre n. Para esto, se definen los conjuntos $A_{w,n}$ como el conjunto formado por todas las cadenas del lenguaje $L_{FULL-SAT}$, que representan fórmulas booleanas de n cláusulas satisfacibles por w y B_w como el conjunto formado por todas las cadenas que representan las cláusulas que son satisfacibles por w. Las cadenas de $A_{w,n}$ están formadas por n concatenaciones de cadenas de B_w separadas por d.

El caso base $n = 1$ se demuestra porque la transducción se realiza solo sobre el primer transductor T_{CLAUSE} que conforma a T_{SAT}, ya que en la cadena de entrada solo hay una d y como se demostró en el Lema 3.1, $T_{CLAUSE}(w)$ contiene todas cláusulas satisfacibles por w. Por tanto $T_{SAT}(wd)$, donde $w \in \{0,1\}^*$, contiene todas las cadenas que representan fórmulas booleanas de una cláusulas satisfacibles por wd.

Una vez demostrado el caso base se asume que el Lema 3.2 es cierto para $n = k$ y se demuestra para $n = k+1$.

El conjunto de todas las fórmulas booleanas con $k+1$ cláusulas, satisfacibles por w, es equivalente al conjunto que forman todas las fórmulas booleanas con k cláusulas, satisfacibles por w, concatenadas con todas las cláusulas satisfacibles por w:

$$A_{w,k+1} = \{xzd \,|\, x \in A_{w,k} \text{ y } z \in B_w\}.$$

Además, por la estructura de T_{SAT} se cumple que el conjunto de todas las cadenas que pertenecen al lenguaje generado por $T_{SAT}((wd)^{k+1})$ es igual al conjunto de todas las cadenas que pertenecen al lenguaje generado por $T_{SAT}((wd)^k)$ concatenadas con todas las cadenas que pertenecen al lenguaje generado por $T_{CLAUSE}(w)$:

$$T_{SAT}((wd)^{k+1}) = \{xzd \,|\, x \in T_{SAT}((wd)^k) \text{ y } z \in T_{CLAUSE}(w)\}.$$

Por hipótesis de inducción se cumple que $A_{w,k} = T_{SAT}((wd)^k)$ y además se cumple que $B_w = T_{CLAUSE}(w)$, lo cual implica que $A_{w,k+1} = T_{SAT}((wd)^{k+1})$, por lo tanto se demuestra el Lema 3.2.

□

A continuación se demuestra el Teorema 3.1.

Demostración del Teorema 3.1. Para la demostrar que el lenguaje generado por la transducción de $L_{0,1,d}$ mediante T_{SAT} es igual a L_{S-SAT}, sea una cadena $r \in L_{0,1,d}$ y sea una cadena $e \in T_{SAT}(r)$, entonces se cumple que la fórmula booleana asociada a e es satisfacible por r, por el Lema 3.2. Ahora sea F una fórmula booleana satisfacible y sea e su cadena asociada, por tanto existe $r \in L_{0,1,d}$ tal que r satisface a F, luego se cumple que $e \in T_{SAT}(r)$, por el Lema 3.2. Por tanto se cumple que:

$$L_{S-SAT} = \{e \,|\, \exists r \in L_{0,1,d} \text{ y } e \in T_{SAT}(r)\}.$$

□

Una consecuencia directa del Teorema 3.1 es que el problema de la palabra de cualquier formalismo que genere el lenguaje $L_{0,1,d}$ y sea cerrado bajo transducción finita, es NP-Duro.

Para demostrar el planteamiento del párrafo anterior, suponga que existe un formalismo G que genera $L_{0,1,d}$, que es cerrado bajo transducción finita.

Sea G' el formalismo que resulta de aplicarle el transductor T_{SAT} a G. Entonces determinar si una cadena e pertenece al lenguaje generado por G', es equivalente a saber si la fórmula booleana a la cual representa e es satisfacible. Por tanto el problema de la palabra para G' es NP-Duro, porque tiene una reducción directa al problema de la satisfacibilidad booleana.

Una restricción importante en la demostración anterior es que la representación de G' tiene que tener tamaño $O(1)$, porque si no el problema de la palabra de G' puede depender además de la cantidad de estados (en el caso de la representación mediante una máquina abstracta) o de la cantidad de producciones, símbolos terminales y no terminales (en el caso de la representación mediante una gramática).

Se conjetura que cualquier formalismo G que genere el lenguaje $L_{0,1,d}$, tiene tamaño $O(1)$ en su representación y como el transductor T_{SAT} tiene una cantidad de estados $O(1)$, entonces G' tiene que tener tamaño $O(1)$ en su representación.

En la literatura consultada, todo formalismo que genera el lenguaje L_{copy} tiene tamaño $O(1)$ en su representación. El lenguaje $L_{0,1,d}$ tiene características similares al lenguaje L_{copy}, razón que apoya la conjetura anterior.

En este capítulo se presentó una estrategia para resolver el SAT usando teoría de lenguajes que se basa en definir y construir el lenguaje de todas las fórmulas satisfacibles. Además se presentó un primer acercamiento para construir este lenguaje, mediante una transducción finita lo cual demuestra que el problema de la palabra para todos los formalismos que generen $L_{0,1,d}$ y sean cerrados bajo transducción finita, es NP-duro. En el próximo capítulo se argumenta que la estrategia presentada en este para construir L_{S-SAT} no es la única, porque se construye una RCG que reconoce el lenguaje L_{S-SAT}, y los lenguajes de concatenación de rango no son cerrados bajo transducción finita.

Capítulo 4

Lenguaje de las fórmulas booleanas satisfacibles empleando gramáticas de concatenación de rango

En este capítulo se presenta cómo resolver el SAT usando el problema de la palabra, mediante gramáticas de concatenación de rango. Para ello se obtiene una gramática de concatenación de tango que reconoce el lenguaje L_{S-SAT}, la cual permite demostrar que las RCG cubren todos los problemas que pertenecen a la clase NP y se deja como problema abierto obtener una RCG que permita reconocer todas las instancias de SAT que son solubles en tiempo polinomial.

Para el desarrollo del capítulo primeramente se describe como reconocer el lenguaje $L_{0,1,d}$ mediante una gramática de concatenación de rango y a continuación se describe por qué no es posible usar las RCG para construir el lenguaje L_{S-SAT} mediante una transducción finita de un lenguaje de concatenación de rango. A pesar de ello, es posible construir una RCG que reconoce todas las fórmulas booleanas satisfacibles y se demuestra que las RCG cubren todos los problemas en la clase NP. Por último se presenta una RCG que permite reconocer problemas 2-SAT, pero que tiene el problema de la palabra no polinomial. Se propone como problema abierto buscar una RCG que permita reconocer problemas 2-SAT en tiempo polinomial.

En la siguiente sección se describe una RCG que reconoce el lenguaje $L_{0,1,d}$.

4.1. $L_{0,1,d}$ como lenguaje de concatenación de rango

En esta sección se presenta una RCG que reconoce el lenguaje

$$L_{0,1,d} = \{(wd)^+ \,|\, w \in \{0,1\}^+\},$$

que puede usarse para representar la asignación de valores a las variables de un SAT. La gramática que reconoce $L_{0,1,d}$ se basa en reconocer una cadena w seguida de un caracter d y después comprobar que las siguientes cadenas sean iguales a w seguidas del caracter d.

Para reconocer $L_{0,1,d}$ se define la gramática $G_{0,1,d}$ como sigue:

$$G_{0,1,d} = (N,T,V,P,S),$$

donde:

- $N = \{S,A,B,C,Eq\}$
- $T = \{0,1,d\}$.
- $V = \{X,Y,P\}$.
- El conjunto de cláusulas P es el siguiente:
    1. $S(X) \to A(X)$
    2. $A(XdY) \to B(Y,X)C(X)$
    3. $B(XdY,P) \to B(Y,P)C(X)Eq(X,P)$
    4. $B(\varepsilon,P) \to \varepsilon$
    5. $C(0X) \to C(X)$
    6. $C(1X) \to C(X)$
    7. $C(\varepsilon) \to \varepsilon$
- El **símbolo inicial** es S.

El predicado Eq se define en [4] y comprueba que dos cadenas sobre un alfabeto sean iguales. Por otro lado, el predicado B recibe un patrón p y una cadena, y determina si algún prefijo de la cadena, que esté seguido de una d, es igual a p. Luego continúa la derivación del resto de la cadena en el predicado B con el mismo patrón. Finalmente la cadena que recibe B es la cadena vacía entonces se deriva en la cadena vacía. Falta por describir el comportamiento del predicado C, que comprueba que las subcadenas intermedias que determina el caracter d, estén formadas por 0 y 1.

A continuación se presenta un ejemplo de cómo $G_{0,1,d}$ reconoce la cadena 101**d**101**d**101**d**.

Las derivaciones que permiten reconocer la cadena 101**d**101**d**101**d** son las siguientes.

1. $S(101\mathbf{d}101\mathbf{d}101\mathbf{d}) \to A(101\mathbf{d}101\mathbf{d}101d)$

2. Cuando se aplica la derivación

$$A(XdY) \rightarrow B(Y, X)C(X)$$

con el vector $[101\mathbf{d}101\mathbf{d}101\mathbf{d}]$, se puede hacer la siguiente sustitución de rango: $X = 101$, $Y = 101\mathbf{d}101\mathbf{d}$, y se tiene que la producción se instanciaría como:

$$A(101\mathbf{d}101\mathbf{d}101\mathbf{d}) \rightarrow B(101\mathbf{d}101\mathbf{d}, 101)C(101).$$

3. Para la producción

$$B(XdY, P) \rightarrow B(Y, P)C(X)Eq(X, P),$$

con el vector $[101\mathbf{d}101\mathbf{d}, 101]$, a las variables X, Y, P se le asignan los valores $X = 101$, $Y = 101\mathbf{d}$, $P = 101$, con lo que se tiene la derivación

$$B(101\mathbf{d}101\mathbf{d}, 101) \rightarrow B(101\mathbf{d}, 101)C(101)Eq(101, 101).$$

4. Cuando se aplica la derivación

$$B(XdY, P) \rightarrow B(Y, P)C(X)Eq(X, P),$$

con el vector $[101\mathbf{d}, 101]$, se puede hacer la siguiente sustitución de rango: $X = 101$, $Y = \varepsilon$, $P = 101$, y se tiene que la producción se instanciaría como:

$$B(101\mathbf{d}, 101) \rightarrow B(\varepsilon, 101)C(101)Eq(101, 101).$$

5. $B(\varepsilon, 101) \rightarrow \varepsilon$

6. $C(101) \rightarrow C(01)$

7. $C(01) \rightarrow C(1)$

8. $C(1) \rightarrow C(\varepsilon)$

9. $C(\varepsilon) \rightarrow \varepsilon$

Como para todos los predicados existe una sustitución de rango en la que estos derivan en la cadena vacía entonces $G_{0,1,d}$ reconoce la cadena $101\mathbf{d}101\mathbf{d}101\mathbf{d}$.

A continuación se demuestra que $G_{0,1,d}$, reconoce el lenguaje $L_{0,1,d}$.

4.1.1. Demostración de que $G_{0,1,d}$ reconoce el lenguaje $L_{0,1,d}$

En esta sección se demuestra que el lenguaje $L_{0,1,d}$ es igual al lenguaje que reconoce la gramática $G_{0,1,d}$, denominado $L_{G_{0,1,d}}$.

Demostración de que $L_{0,1,d} = L_{G_{0,1,d}}$. Para demostrar que $L_{0,1,d} = L_{G_{0,1,d}}$, se debe demostrar que $L_{G_{0,1,d}}$ es subconjunto de $L_{0,1,d}$ y que $L_{0,1,d}$ es subconjunto de $L_{G_{0,1,d}}$.

Para demostrar que $L_{G_{0,1,d}} \subseteq L_{0,1,d}$, sea una cadena $r \in L_{G_{0,1,d}}$, por lo tanto existen las cadenas z_r y w, tal que existe una secuencia de derivaciones desde $S(r)$ hasta $B(z_r, w)$ y desde $B(z_r, w)$ hasta la cadena vacía.

Del predicado S solo se deriva al predicado A y de A se deriva a B en la cláusula $A(XdY) \to B(Y,X)C(X)$. Cuando la sustitución de rango identifica un caracter d en la cadena de entrada y asocia la subcadena izquierda a la variable X y la subcadena derecha a la variable Y. Por la estructura de la gramática el predicado C reconoce una cadena si y solo si esta cadena está formada solamente por 0 y 1. En la derivación de la cláusula $A(XdY) \to B(Y,X)C(X)$ se cumple que $X = z_r$ e $Y = w$, y como C reconoce la cadena w, entonces se cumple que w es una cadena binaria y z_r es igual a la cadena de entrada sin su primer bloque de 0 y 1.

Por la estructura de la cláusula $B(XdY,P) \to B(Y,P)C(X)Eq(X,P)$ solo se puede hacer una derivación en esta cláusula si a las variables X y Y se le asocian subcadenas de la primera cadena del vector que reconoce B, tales que el valor de X es una cadena binaria y además es igual a la segunda cadena del vector que reconoce B. Si esta sustitución de rango existe, entonces se vuelve a derivar e B pero esta vez el vector en entrada está formado por la primera cadena del vector de entrada original sin su primer bloque de unos y la segunda cadena del vector de entrada original.

En cada derivación del predicado B se comprueba que la primera cadena del vector de entrada está formada por un prefijo w', tal que w', esta seguida de una d y w' es igual a la segunda cadena del vector de entrada. Además se comprueba que el resto de la primera cadena del vector de entrada es la cadena vacía o cumple la misma condición.

Del predicado B solo se deriva a la cadena vacía si la primera cadena del vector de entrada es la cadena vacía. Por tanto el predicado B reconoce el vector $[z_r, w]$ si z_r está formado por varios bloques de 0 y 1, separados por una d y cada uno de estos bloques es igual a w.

Entonces como B reconoce a $[z_r, w]$, se cumple que $z_r = w^n$, donde $n \in \mathbb{N}$. Como $r = wdz_r$ y $z_r = w^n$, esto implica que $r = w^{n+1}$, por lo que $r \in L_{0,1,d}$, por tanto $L_{G_{0,1,d}} \subseteq L_{0,1,d}$.

Para demostrar que $L_{0,1,d} \subseteq L_{G_{0,1,d}}$, sea una cadena $(wd)^n \in L_{0,1,d}$, donde $n \in \mathbb{N}$ y w es una cadena binaria. En la cláusula $A(XdY) \to B(Y,X)C(X)$, se puede hacer la sustitución de rango de manera que $X = w$ e $Y = (wd)^{n-1}$, como w es una cadena binaria se cumple que C reconoce la cadena w y entonces se deriva en el predicado B, con el vector $[(wd)^{n-1}, w]$.

Si se deriva en el predicado B con el vector $[(wd)^k, w]$, donde $k \in \mathbb{N} \cup \{0\}$ y w es una cadena binaria, si $k \neq 0$ se toma la cláusula $B(XdY, P) \to B(Y, P)C(X)Eq(X, P)$ y se puede realizar la sustitución de rango de manera que $X = w$ e $Y = (wd)^{k-1}$. Se cumple que Eq reconoce el vector $[w, w]$ y C reconoce la cadena w, por lo que se vuelve a derivar en el predicado B con el vector $[(wd)^{k-1}, w]$. Si $k = 0$ se deriva en la cadena vacía.

Por tanto B reconoce el vector $[(wd)^{n-1}, w]$, por lo que $(wd)^n$ se reconoce por $G_{0,1,d}$. Luego $(wd)^n \in L_{G_{0,1,d}}$, entonces se cumple que $L_{0,1,d} \subseteq L_{G_{0,1,d}}$, lo cual implica que $L_{0,1,d} = L_{G_{0,1,d}}$. □

En la próxima sección se analiza por qué no es posible construir L_{S-SAT} mediante una transducción finita usando una RCG.

4.1.2. Imposibilidad de construir L_{S-SAT} mediante una transducción finita usando una RCG

Como demostró en la sección anterior, el lenguaje $L_{0,1,d}$ se reconoce mediante una RCG. Con la idea expuesta en las secciones del capítulo 3, donde se muestra cómo construir L_{S-SAT} mediante una transducción finita, se puede usar $G_{0,1,d}$ como formalismo para generar $L_{0,1,d}$. Esto no es posible porque las RCG no son cerradas bajo transducción finita como se mencionó en el capitulo 2.

A continuación se muestra una RCG que reconoce el lenguaje L_{S-SAT}, lo cual demuestra que la transducción finita no es la única vía para construir este lenguaje.

4.2. Construir L_{S-SAT} mediante una RCG

En esta sección se presenta una RCG que reconoce las fórmulas booleanas satisfacibles. Esto permite demostrar que las RCG cubren todos los problemas de la clase NP.

En la construcción de L_{S-SAT} mediante una transducción finita se usa el lenguaje $L_{0,1,d}$ para generar todas las posibles interpretaciones de cualquier fórmula booleana y a partir de estas interpretaciones el transductor T_{S-SAT} genera todas las posibles fórmulas booleanas satisfacibles para cada una.

En esta sección, a diferencia de la estrategia seguida en el capitulo 3, se presenta una RCG que reconoce las fórmulas booleanas satisfacibles. La cual durante el reconocimiento de la cadena de entrada, genera todas las posibles interpretaciones de la fórmula booleana que satisfacen la primera cláusula y luego comprueba si alguna de ellas satisface el resto de las cláusulas.

Para definir la gramática, se agrupan las cláusulas en 4 fases, en dependencia de las tareas que cumplen durante el reconocimiento.

- **Primera fase:** representa la derivación inicial de la gramática.
- **Segunda fase:** se encarga de generar todas las posibles interpretaciones de las variables, que satisfacen la primera cláusula. En esta fase se definen 2 estados: positivo (significa que la cadena de 0 y 1 generada ya satisface la primera cláusula) y negativo (significa que la cadena de 0 y 1 generada aún no satisface la primera cláusula). Estos estados se representan por los predicados P y N, respectivamente.
- **Tercera fase:** comprueba que la interpretación que se define en la fase anterior sea satisfacible para el resto de las cláusulas.
- **Cuarta fase:** define el algoritmo para determinar si una interpretación satisface una cláusula dada. En esta fase se definen 2 estados: positivo (significa que la interpretación ya satisface la cláusula actual) y negativo (significa que la interpretación aún no satisface la cláusula actual). Estos estados se representan por los predicados Cp y Cn respectivamente.

Seguidamente, se define la siguiente RCG que reconoce el lenguaje L_{S-SAT}:

$$G_{S-SAT} = (N,T,V,P,S),$$

donde:

- $N = \{S,A,B,C,P,N,Cp,Cn\}$
- $T = \{a,b,c,d\}$.
- $V = \{X,Y,X_1,X_2\}$.
- El **símbolo inicial** es S.

A continuación se desglosa el conjunto de **cláusulas** P de acuerdo a las fases descritas.

- **Primera fase:** Representa la cláusula de derivación inicial de la gramática:
  1. $S(X) \to A(X)$.
- **Segunda fase:** El siguiente conjunto de cláusulas genera una cadena de 0 y 1 que asigna valores a las variables de la fórmula booleana, de forma que satisfagan la primera cláusula.

2. $A(aX) \to P(X,1)$
3. $A(aX) \to N(X,0)$
4. $A(bX) \to N(X,1)$
5. $A(bX) \to P(X,0)$
6. $A(cX) \to N(X,1)$
7. $A(cX) \to N(X,0)$
8. $P(aX,Y) \to P(X,Y1)$
9. $P(aX,Y) \to P(X,Y0)$
10. $P(bX,Y) \to P(X,Y1)$
11. $P(bX,Y) \to P(X,Y0)$
12. $P(cX,Y) \to P(X,Y1)$
13. $P(cX,Y) \to P(X,Y0)$
14. $P(dX,Y) \to B(X,Y)$
15. $N(aX,Y) \to P(X,Y1)$
16. $N(aX,Y) \to N(X,Y0)$
17. $N(bX,Y) \to N(X,Y1)$
18. $N(bX,Y) \to P(X,Y0)$
19. $N(cX,Y) \to N(X,Y1)$
20. $N(cX,Y) \to N(X,Y0)$

El no terminal A representa el predicado por donde inician las derivaciones de esta fase, P representa que con los valores de las variables que se han generado, la cláusula ya tiene un valor de verdad positivo y N representa que con esos mismos valores la fórmula booleana aún tiene un valor de verdad negativo.

Del no terminal A se deriva a los predicados P y N en dependencia del valor asignado a la variable del literal que se encuentra al inicio del rango actual. El predicado P deriva hacia sí mismo independientemente del símbolo, exceptuando el símbolo d, caso en el que se deriva en B y se procede a la siguiente fase.

Por último del no terminal N se deriva a los predicados P y N en dependencia del valor asignado a la variable del literal que se encuentra al inicio del rango actual.

- **Tercera fase:** El siguiente conjunto de cláusulas comprueba que la asignación de variables que se realiza en la fase anterior sea verdadera para las restantes cláusulas.

21. $B(X_1dX_2,Y) \to C(X_1,Y)B(X_2,Y)$
22. $B(\varepsilon,Y) \to \varepsilon$

El predicado B permite identificar las cláusulas restantes, mientras que el predicado C comprueba que cada cláusula identificada por el predicado B sea satisfacible con los valores de las variables que recibe en su segundo argumento. Este comportamiento se define en la cuarta fase.

- **Cuarta fase:** En esta fase se define el comportamiento de C, que recibe una cláusula y una interpretación de las variables y comprueba que dicha interpretación sea verdadera para la cláusula analizada.

23. $C(X,Y) \to Cn(X,Y)$
24. $Cn(aX,1Y) \to Cp(X,Y)$
25. $Cn(aX,0Y) \to Cn(X,Y)$
26. $Cn(bX,1Y) \to Cn(X,Y)$
27. $Cn(bX,0Y) \to Cp(X,Y)$
28. $Cn(cX,1Y) \to Cn(X,Y)$
29. $Cn(cX,0Y) \to Cn(X,Y)$
30. $Cp(aX,1Y) \to Cp(X,Y)$
31. $Cp(aX,0Y) \to Cp(X,Y)$
32. $Cp(bX,1Y) \to Cp(X,Y)$
33. $Cp(bX,0Y) \to Cp(X,Y)$
34. $Cp(cX,1Y) \to Cp(X,Y)$
35. $Cp(cX,0Y) \to Cp(X,Y)$
36. $Cp(\varepsilon,\varepsilon) \to \varepsilon$

Observe que este funcionamiento sigue la misma idea que el descrito en la segunda fase: tiene un predicado que representa un estado positivo (Cp) y un predicado que representa un estado positivo (Cn). La diferencia es que no se genera la cadena sino que se comprueba con el patrón que se construye en la segunda fase y que cada uno de los no terminales de esta fase recibe como argumento.

A continuación se demuestra que el lenguaje que reconoce G_{S-SAT} es exactamente igual al lenguaje que representa todas las fórmulas booleanas satisfacibles descritas mediante el lenguaje $L_{FULL-SAT}$.

4.3. Demostración de que la gramática G_{S-SAT} reconoce el lenguaje L_{S-SAT}

En esta sección se demuestra que G_{S-SAT} reconoce el lenguaje L_{S-SAT}.

Teorema 4.1. *Dada una cadena $e \in L_{FULL-SAT}$, G_{S-SAT} reconoce la cadena e si y solo si la fórmula booleana asociada a e es satisfacible.*

A continuación, se presentan algunas definiciones que serán usadas en la demostración del Teorema 4.1.

Definición 4.1. *Sea una cadena $w \in \{0,1\}^*$, los valores de w se le asignan a una fórmula booleana F, cuando w tiene la misma longitud que el número de variables de F. Además si el i-ésimo caracter de w es un 1 a la i-ésima variable de F se le asigna el valor true y si el i-ésimo caracter de w es un 0 a la i-ésima variable de F se le asigna el valor false.*

Definición 4.2. *Una cadena binaria w satisface una fórmula booleana F si al asignarle los valores de w a F, se obtiene un valor de verdad positivo.*

Definición 4.3. *Una cadena binaria w satisface a una cadena $e \in L_{FULL-SAT}$ si w satisface la fórmula booleana asociada a e.*

Para la demostración del Teorema 4.1 se usarán los siguientes lemas.

Lema 4.1. *El predicado C de la gramática G_{S-SAT}, reconoce las cadenas $q \in \{a,b,c\}^+$ y $w \in \{0,1\}^*$ si y solo si w satisface a la cláusula que representa q.*

Lema 4.2. *El predicado B de la gramática G_{S-SAT}, reconoce las cadenas $e \in L_{FULL-SAT}$ y $w \in \{0,1\}^*$ si y solo si w satisface a todas las cláusulas de e.*

Lema 4.3. *Dada una cadena $e \in L_{FULL-SAT}$, el conjunto de cadenas W formado por todas las cadenas $w \in \{0,1\}^*$ tales que existe una secuencia de derivaciones desde del predicado $A(e)$ hasta $B(z_e, w)$, donde z_e es igual a la cadena e sin su última cláusula, es exactamente igual a al conjunto de todas las interpretaciones que hacen verdadera la primera cláusula de e.*

La idea de la demostración del Teorema 4.1 es probar que dadas una cadena $q \in \{a,b,c\}^+$ y $w \in \{0,1\}^*$, $C(q,w)$ se reconoce por la gramática si y solo si w satisface la cláusula que representa q, Lema 4.1. Después, dadas una cadena $e \in L_{FULL-SAT}$ y $w \in \{0,1\}^*$, hacer una inducción sobre la cantidad de cláusulas de e para demostrar que G_{S-SAT} reconoce $B(e,w)$ si y solo si w satisface todas las cláusulas de e, Lema 4.2. Posteriormente, probar que durante la segunda fase se generan todas las cadenas binarias que satisfacen la primera cláusula de la cadena de entrada, Lema 4.3. Para terminar, probar que el lenguaje que reconoce G_{S-SAT} es igual a L_{S-SAT}.

A continuación se demuestra el Lema 4.1.

Demostración del Lema 4.1. Sean las cadenas $q \in \{a,b,c\}^+$ y $w \in \{0,1\}^*$, asociadas a una cláusula y a una interpretación de variables respectivamente, el predicado $C(q,w)$ y la cláusula asociada a q, F_q.

Para demostrar que $C(q,w)$ se reconoce por G_{S-SAT} si F_q es satisfacible por w, suponga que F_q es satisfacible por w, entonces se debe demostrar que existe una secuencia de derivaciones desde $C(q,w)$ hasta la cadena vacía.

Como w satisface a F_q, cuando se le asignan los valores de w a las variables de F_q existe una variable sin negar con valor *true* o una variable negada con valor *false*.

Del predicado C se deriva al predicado Cn, en la fase 4. Las únicas derivaciones de la gramática donde se deriva del predicado Cn a Cp es la combinación de una a y un 1 o de una b y un 0 y como w satisface F_q esta combinación existe.

Por último, Cp siempre deriva en sí mismo o en la cadena vacía por lo que queda demostrado que existe una secuencia de derivaciones desde $C(q,w)$ hasta la cadena vacía.

Para finalizar la demostración del Lema 4.1, es necesario probar que si $C(q,w)$ se reconoce entonces w satisface a F_q. Por la estructura de la gramática, si existe una secuencia de derivaciones desde $C(q,w)$ hasta la cadena vacía entonces hay una derivación desde Cn hacia Cp, sin pérdida de la generalidad esta derivación ocurre en el en i-ésimo caracter de q. Esta derivación solo es posible por una combinación de una a y un 1 o de una b y un 0, por lo tanto una de estas combinaciones existe. Por lo que cuando se le asignan los valores de w a las variables de F_q la i-ésima variable está sin negar con valor *true* o está negada con valor *false*, lo cual implica que w satisface F_q, por tanto se demuestra el Lema 4.1. □

Seguidamente se demuestra el Lema 4.2.

Demostración del Lema 4.2. Sean las cadenas e y w, asociadas a una fórmula booleana y a una interpretación de variables respectivamente, el predicado $B(e,w)$ y la cláusula $B(X_1dX_2,Y) \to C(X_1,Y)B(X_2,Y)$.

Para demostrar el Lema 4.2 se hará una inducción sobre la cantidad de cláusulas n de la fórmula booleana asociada a e.

Para $n=1$ se cumple que, al realizar la sustitución de rango, los rangos asociados a las variables X_1 y X_2 son e sin su último caracter y la cadena vacía respectivamente. Por tanto $B(e,w)$ se reconoce por la gramática si y solo si $C(X_1,w)$ se reconoce, porque $B(\varepsilon,w)$ deriva en la cadena vacía. $C(X_1,w)$ se reconoce si y solo si w satisface a X_1, Lema 4.1, por lo que se demuestra el caso base.

Una vez demostrado el caso base se asume que el Lema 4.2 es cierto para $n=k$ y se demuestra para $k+1$.

En todas las posibles sustituciones de rango de X_1 y X_2, $C(X_1,w)$ solo se reconoce si $|X_1|=|w|$, por lo tanto, el caso de sustitución de rango que ocupa a la demostración es cuando $|X_1|=|w|$, porque para el resto de las sustituciones de rango $C(X_1,w)$ no se reconoce por la gramática.

La cadena w satisface todas las cláusulas de e si y solo si satisface a la primera cláusula de e y el resto de las cláusulas de e, que en este caso están asociadas a las variables X_1 y X_2 respectivamente. Precisamente $B(e,w)$ se reconoce si y solo si se reconoce $C(X_1,w)$ y $B(X_2,w)$. $C(X_1,w)$ se reconoce si y solo si w satisface a X_1, por el Lema 4.1 y $B(X_2,w)$ se reconoce si y solo si w satisface todas las cláusulas de X_2 por hipótesis de inducción, ya que X_2 tiene k cláusulas. Por tanto se demuestra el Lema 4.2. □

A continuación se demuestra el Lema 4.3.

Demostración del Lema 4.3. Sea la cadena e asociada a una fórmula booleana y el predicado $A(e)$, para demostrar el Lema 4.3 se utilizan 2 conjuntos: W y W'. W representa el conjunto de todas las cadenas de 0 y 1 que satisfacen la primera cláusula de la fórmula asociada a e , denominada F_{1q}, y W', representa el conjunto de todas las cadenas w' tales que existe una secuencia de derivaciones desde $A(e)$ hasta $B(z_e,w')$, donde z_e está conformada por todas las cláusulas de e menos la primera.

Dadas estas definiciones, para probar el Lema 4.3, es necesario demostrar que $W = W'$, para ello se debe demostrar que W es subconjunto de W' y W' es subconjunto de W.

Para demostrar que $W \subseteq W'$, se toma una cadena w tal que $w \in W$, es decir, w satisface a F_{1q}. Por tanto en F_{1q}, cuando se le asignan los valores de w a las variables de F_{1q}, existe una variable sin negar con valor *true* o una variable negada con valor *false*, lo que representa una combinación de una a y un 1 o de una b y un 0 en una de las derivaciones de la segunda fase.

Por la estructura de la gramática del predicado A, solo hay derivaciones hacia P con una de estas 2 combinaciones, el resto son hacia el predicado N y del predicado N solo hay derivaciones a P con una de las combinaciones anteriores. Por tanto, como existe una combinación de una a y un 1 o de una b y un 0, existe una secuencia de derivaciones que lleva del predicado A al predicado P pasando por N o sin pasar por N. Como el predicado P solo tiene derivaciones hacia sí mismo o hacia $B(z_e, w)$ se cumple que $w \in W'$.

Para demostrar que $W' \subseteq W$, se toma una cadena w' tal que $w' \in W'$, por lo que existe una secuencia de derivaciones desde $A(e)$ a $B(z_e, w')$. Por la estructura de la gramática solo se puede derivar al predicado B desde el predicado P, y a su vez de este predicado solo se puede derivar mediante una combinación de una a y un 1 o de una b y un 0 en la gramática. Por tanto en F_{1q}, cuando se le asignan los valores de w' a las variables de F_{1q}, existe una variable sin negar con valor *true* o una variable negada con valor *false*. Entonces se cumple que w' satisface a F_{1q} por lo que $w' \in W$. Con esto se demuestra que $W' = W$, por tanto se cumple el Lema 4.3. $\square$

Seguidamente se demuestra el Teorema 4.1.

Demostración del Teorema 4.1. Sea la cadena e asociada a una fórmula booleana y el predicado $S(e)$, para demostrar que el lenguaje que reconoce G_{S-SAT} es exactamente igual al lenguaje que representa todas las fórmulas booleanas satisfacibles se define el lenguaje $L_{G_{S-SAT}}$ que representa el lenguaje de todas las cadenas que se reconocen por G_{S-SAT}. Entonces es necesario demostrar que $L_{S-SAT} = L_{G_{S-SAT}}$.

Para demostrar que $L_{S-SAT} \subseteq L_{G_{S-SAT}}$, sea una fórmula booleana satisfacible F y sea e su representación como cadena en el lenguaje $L_{FULL-SAT}$, entonces existe una cadena binaria w, con longitud igual a la cantidad de variables de F, que satisface a F.

Como w satisface a F entonces w pertenece al conjunto de cadenas que satisfacen a la primera cláusula de F. Por el Lema 4.3 existe una secuencia de derivaciones desde el predicado $S(e)$ hasta $B(z_e, w)$, y como w satisface todas las cláusulas de F entonces $B(z_e, w)$ deriva en la cadena vacía, por el Lema 4.2, por lo que e se reconoce por G_{S-SAT}, lo cual implica que $L_{S-SAT} \subseteq L_{G_{S-SAT}}$.

Para completar la demostración es necesario demostrar que $L_{G_{S-SAT}} \subseteq L_{S-SAT}$. Sea una cadena e que se reconoce por G_{S-SAT} y sea F la fórmula booleana asociada a e, entonces existe una cadena binaria w tal que existe una secuencia de derivaciones desde $A(e)$ a $B(z_e, w)$ y de $B(z_e, w)$ a la cadena vacía, por tanto w satisface a la primera cláusula de F y a las restantes también, por el Lema 4.3 y 4.2 respectivamente, luego w satisface a F, por lo que F es satisfacible. Esto implica que $L_{G_{S-SAT}} \subseteq L_{S-SAT}$ y con esto se demuestra que $L_{G_{S-SAT}} = L_{S-SAT}$. □

En la siguiente sección se presenta un ejemplo del reconocimiento de una cadena por G_{S-SAT}.

4.3.1. Ejemplo de reconocimiento de G_{S-SAT}

En esta sección se presentan 2 ejemplos del funcionamiento de G_{S-SAT} en el primero se muestra cómo se reconoce la cadena asociada a la fórmula booleana $(x_1 \vee x_2) \wedge (x_1) \wedge (\neg x_2)$ y en el segundo se muestra cómo no se reconoce la fórmula booleana asociada a $x_1 \wedge \neg x_1$.

La cadena asociada a $(x_1 \vee x_2) \wedge (x_1) \wedge (\neg x_2)$ es $aa\mathbf{d}ac\mathbf{d}cb\mathbf{d}$ y una posible secuencia de derivaciones asociada a esta cadena en G_{S-SAT} es la siguiente:

1. $S(aa\mathbf{d}ac\mathbf{d}cb\mathbf{d}) \to A(aa\mathbf{d}ac\mathbf{d}cb\mathbf{d})$
2. $A(aa\mathbf{d}ac\mathbf{d}cb\mathbf{d}) \to P(a\mathbf{d}ac\mathbf{d}cb\mathbf{d}, 1)$
3. $P(a\mathbf{d}ac\mathbf{d}cb\mathbf{d}, 1) \to P(\mathbf{d}ac\mathbf{d}cb\mathbf{d}, 10)$
4. $P(\mathbf{d}ac\mathbf{d}cb\mathbf{d}, 10) \to B(ac\mathbf{d}cb\mathbf{d}, 10)$
5. Para la producción
$$B(X_1 dX_2, Y) \to C(X_1, Y)B(X_2, Y),$$
con el vector $[ac\mathbf{d}cb\mathbf{d}, 10]$, se le asignan los valores $X_1 = ac$, $X_2 = cb\mathbf{d}$, $Y = 10$, con lo que se tiene la derivación:
$$B(ac\mathbf{d}cb\mathbf{d}, 10) \to C(ac, 10)B(cb\mathbf{d}, 10)$$
6. Para la producción
$$B(X_1 dX_2, Y) \to C(X_1, Y)B(X_2, Y),$$
con el vector $[cb\mathbf{d}, 10]$, se le asignan los valores $X_1 = cb$, $X_2 = \varepsilon$, $Y = 10$, con lo que se tiene la derivación:
$$B(cb\mathbf{d}, 10) \to C(cb, 10)B(\varepsilon, 10)$$

7. $B(\varepsilon, 10) \to \varepsilon$
8. $C(ac, 10) \to Cn(ac, 10)$
9. $Cn(ac, 10) \to Cp(c, 0)$
10. $Cp(c, 0) \to Cp(\varepsilon, \varepsilon)$
11. $Cp(\varepsilon, \varepsilon) \to \varepsilon$
12. $C(cb, 10) \to Cn(cb, 10)$
13. $Cn(cb, 10) \to Cn(b, 0)$
14. $Cn(b, 0) \to Cp(\varepsilon, \varepsilon)$
15. $Cp(\varepsilon, \varepsilon) \to \varepsilon$

Como en todas las instancias de los no terminales existe una sustitución de rango que provoca que todos los predicados deriven en la cadena vacía, entonces $aa\mathbf{d}ac\mathbf{d}cb\mathbf{d}$ se reconoce por G_{S-SAT}. Esto coincide con el hecho de que $(x_1 \vee x_2) \wedge (x_1) \wedge (\neg x_2)$ es satisfacible, para la asignación de valores $x_1 = 1$ y $x_2 = 0$.

Seguidamente se presenta una cadena que representa una fórmula que no es satisfacible y por tanto la cadena asociada a dicha fórmula no se reconoce por G_{S-SAT}. La cadena asociada a $x_1 \wedge \neg x_1$ es $a\mathbf{d}b\mathbf{d}$, la secuencia de derivaciones asociada a esta cadena en G_{S-SAT} es la siguiente:

1. $S(a\mathbf{d}b\mathbf{d}) \to A(a\mathbf{d}b\mathbf{d})$
2. $A(a\mathbf{d}b\mathbf{d}) \to P(\mathbf{d}b\mathbf{d}, 1)$
3. $A(a\mathbf{d}b\mathbf{d}) \to N(\mathbf{d}b\mathbf{d}, 0)$

En la anterior secuencia de derivaciones hay 2 caminos posibles, cuando $A(a\mathbf{d}b\mathbf{d})$ deriva como $P(\mathbf{d}b\mathbf{d}, 1)$ o cuando $A(a\mathbf{d}b\mathbf{d})$ deriva como $N(\mathbf{d}b\mathbf{d}, 0)$. La siguiente secuencia de derivaciones corresponda al predicado P con el vector $[\mathbf{d}b\mathbf{d}, 1]$:

2. $P(\mathbf{d}b\mathbf{d}, 1) \to B(b\mathbf{d}, 1)$
3. Para la producción

$$B(X_1 dX_2, Y) \to C(X_1, Y)B(X_2, Y),$$

con el vector $[b\mathbf{d}, 1]$, la única asignación de valores posibles es: $X_1 = b$, $X_2 = \varepsilon$, $Y = 1$, con lo que se tiene la derivación:

$$B(b\mathbf{d}, 1) \to C(b, 1)B(\varepsilon, 1)$$

4. $B(\varepsilon,1) \rightarrow \varepsilon$
5. $C(b,1) \rightarrow Cn(b,1)$
6. $Cn(b,1) \rightarrow Cn(\varepsilon,\varepsilon)$

Cuando se realizan todas las posibles derivaciones para el predicado P con el vector $[\mathbf{d}b\mathbf{d},1]$, no se logra derivar en la cadena vacía, por lo tanto el predicado P no reconoce el vector $[\mathbf{d}b\mathbf{d},1]$.

Para el predicado N, con el vector $[\mathbf{d}b\mathbf{d},0]$, no existe ninguna derivación posible, por lo que N no reconoce el vector $[\mathbf{d}b\mathbf{d},0]$.

Entonces después de realizarse todas las posibles derivaciones y sustituciones de rango, G_{S-SAT} no reconoce la cadena $a\mathbf{d}b\mathbf{d}$. Esto coincide con el hecho de que $x_1 \wedge \neg x_1$ no es satisfacible para ninguna asignación de valores a sus variables.

Como G_{S-SAT} reconoce las fórmulas booleanas satisfacibles, para determinar si una fórmula es satisfacible se debe determinar si su cadena asociada se reconoce por G_{S-SAT}, por lo que es necesario analizar la complejidad del problema de la palabra para G_{S-SAT}.

4.3.2. Análisis de la complejidad computacional del reconocimiento en G_{S-SAT}

Como se mencionó en el capítulo 2 no todas las RCG tienen un algoritmo de reconocimiento polinomial y G_{S-SAT} es un ejemplo de ello.

La fase de la gramática que provoca que le algoritmo de reconocimiento sea exponencial es la segunda, ya que se generan todas las cadenas binarias que representan la asignación de valores para las variables booleanas. Estas cadenas se pasan como argumentos a los predicados de fases posteriores, mediante las derivaciones de la gramática.

Si se analiza el algoritmo de reconocimiento descrito en [4] un factor en la complejidad del algoritmo de reconocimiento es la cantidad de rangos posibles para una cadena que se reconoce por un predicado. En este caso la cadena que representa los valores de las variables de la fórmula booleana puede tomar 2^n valores distintos, donde n es la cantidad de variables en la fórmula booleana, ya que dicha cadena se genera durante la segunda fase, donde la gramática es ambigua y en cada derivación hay decisiones que generan valores distintos.

Como se pueden generar 2^n cadenas, cada cadena tiene n^2 rangos y cada cadena generada se pasa como argumento a algún predicado de la gramática, la cantidad de rangos totales que se deben analizar durante el proceso de reconocimiento sería $n^2 2^n$.

Aunque esta es una cota burda ya que para cada cadena no se utilizan todos los posibles rangos en el proceso de derivación de la gramática, por lo que la complejidad es mucho menor pero sigue siendo exponencial con respecto al tamaño de la cadena de entrada.

El resto de las fases de la gramática tienen una complejidad de m^2 donde m es la cantidad de caracteres en la cadena de entrada, por lo que la complejidad total sería $O(2^n m^2)$.

En la siguiente sección se presentan algunos resultados que se derivan de la gramática G_{S-SAT}.

4.3.3. Resultados de la gramática G_{S-SAT}

La gramática G_{S-SAT} demuestra que no es necesario que un formalismo sea cerrado bajo transducción finita para construir el lenguaje de todas las fórmulas booleanas satisfacibles. Además, se obtiene una gramática que tiene tamaño $O(1)$. Este resultado apoya la conjetura formulada en el capítulo 3, que establece que todo formalismo que sea capaz de describir el lenguaje $L_{0,1,d}$, tiene un tamaño $O(1)$ en su representación.

En [3] se menciona las RCG cubren todos los problemas de la clase P. Como se mostró en la sección anterior con la gramática G_{S-SAT}, existe una RCG que reconoce el lenguaje de las fórmulas satisfacibles, y como el SAT se puede reducir a cualquier problema en NP en una complejidad polinomial, entonces para todo problema en NP también existe una RCG que lo reconoce en su representación como lenguaje formal.

En la próxima sección se utilizan las gramáticas de concatenación de rango para determinar si una fórmula booleana asociada a una de las instancias polinomiales del SAT, es satisfacible.

4.4. Instancias de SAT polinomiales empleando RCG

En esta sección se presenta una RCG que es capaz de reconocer problemas SAT, satisfacibles que pertenecen al 2-SAT, es decir, problemas SAT donde cada cláusula tiene a lo sumo 2 literales. La idea detrás de esta gramática es obtener una RCG que reconozca cuando la fórmula booleana pertenece al conjunto de fórmulas booleanas de 2-SAT y luego intersectar dicha gramática con G_{S-SAT}. Para ello se define la siguiente RCG:

$$G_{2-SAT} = (N, T, V, P, S),$$

donde:

- $N = \{S, A, A_0, A_1, A_2, A_3\}$
- $T = \{a, b, c, d\}$.
- $V = \{X, Y, X_1, X_2\}$.

- El conjunto de cláusulas P es el siguiente:

1. $S(X) \rightarrow A(X)$
2. $A(X_1dX_2) \rightarrow A_0(X_1)A(X_2)$
3. $A(\varepsilon) \rightarrow \varepsilon$
4. $A_0(aX) \rightarrow A_1(X)$
5. $A_0(bX) \rightarrow A_1(X)$
6. $A_0(cX) \rightarrow A_0(X)$
7. $A_1(aX) \rightarrow A_2(X)$
8. $A_1(bX) \rightarrow A_2(X)$
9. $A_1(cX) \rightarrow A_1(X)$
10. $A_2(aX) \rightarrow A_3(X)$
11. $A_2(bX) \rightarrow A_3(X)$
12. $A_2(cX) \rightarrow A_2(X)$
13. $A_2(\varepsilon) \rightarrow \varepsilon$

- El **símbolo inicial** es S.

El funcionamiento de la gramática anterior es el siguiente: la segunda cláusula permite reconocer todas las cláusulas asociadas a la cadena original. Las cláusulas de la 4 a la 13 permiten contar la cantidad de a o b en una cláusula, o sea, la cantidad de literales de cada cláusula. Para esto se definen 4 estados: A_0, A_1, A_2 y A_3. A_0 representa que se reconocieron 0 a o b, A_1 representa que se reconocieron una a o b, A_2 representa que se reconocieron 2 a o b y A_3 representa que se reconocen más de 2 a o b.

Si se observa el enfoque seguido en la construcción de G_{S-SAT}, en la representación del SAT como cadena se trabaja con una instancia del SAT general. Por lo que no se tienen en cuenta las propiedades específicas del problema, que en el caso de las instancias polinomiales, es lo que permite que el algoritmo para las mismas sea polinomial.

Finalmente la gramática que reconoce los problemas 2-SAT satisfacibles sería:

$$G_{S-2-SAT} = G_{S-SAT} \cap G_{2-SAT}.$$

Sin embargo, el problema de la palabra para $G_{S-2-SAT}$ es exponencial y se conoce que para el 2-SAT existe un algoritmo polinomial.

Seguidamente se presentan algunos problemas, los cuales se plantean como problemas abiertos para futuras investigaciones.

4.5. Problemas abiertos propuestos

En esta sección se proponen varios problemas abiertos, los cuales tienen relación con el contenido de las secciones de este capítulo.

Como se demostró en la sección 4.1.1, la gramática $G_{0,1,d}$ reconoce el lenguaje $L_{0,1,d}$, pero las RCG no son cerradas bajo transducción finita. Por tanto cuando se aplica T_{SAT} a $G_{0,1,d}$, el formalismo resultante no es necesariamente una RCG, es posible que el formalismo resultante sea la gramática G_{S-SAT} o una gramática equivalente. Se propone como problema abierto, analizar qué tipo de formalismo se obtiene al aplicar el transductor T_{SAT} sobre $G_{0,1,d}$ y realizar un análisis de la complejidad del problema de la palabra para dicho formalismo.

Otro aspecto a considerar sería investigar qué propiedades de las RCG limitan que estas no sean cerradas bajo transducción finita. Una vez identificadas estas propiedades se puede tratar de construir un nuevo formalismo basado en las RCG que sea cerrado bajo transducción finita y comprobar si este formalismo es capaz de describir el lenguaje $L_{0,1,d}$.

Por otro lado, analizando el contenido de la sección 4.4, como las RCG cubren todos los problemas de la clase P, entonces es posible diseñar una RCG para el 2-SAT y para cada instancia polinomial del SAT, donde el problema de la palabra sea polinomial para dicha gramática. Lo anterior queda propuesto como un problema abierto, porque tiene que existir una RCG que reconozca el 2-SAT en tiempo polinomial. Lo cual puede conducir a encontrar nuevas instancias polinomiales del SAT.

En este capítulo se construyó el lenguaje L_{S-SAT}, mediante una RCG, lo que demuestra que no es necesario el transductor T_{SAT} para construir L_{S-SAT}.

Por otro lado G_{S-SAT} demuestra que todos los problemas en NP se reconocen por una RCG y se presenta un primer acercamiento para describir los problemas SAT polinomiales mediante un RCG, dejando abierto el problema de encontrar una RCG que permita reconocer el 2-SAT y el problema de la palabra para esta RCG sea polinomial.

En el próximo capítulo se analizan las conclusiones del trabajo.

Conclusiones

En este trabajo se presentó una estrategia para resolver el SAT usando teoría de lenguajes, la cual se basa en definir una codificación de una fórmula booleana en una cadena y definir y construir el lenguaje de todas las fórmulas booleanas satisfacibles L_{S-SAT}. Luego para determinar si una fórmula booleana es satisfacible es necesario verificar si la cadena asociada a dicha fórmula pertenece a L_{S-SAT}.

En el capítulo 3, se construyó L_{S-SAT} mediante el transductor finito T_{SAT} que recibe cadenas del lenguaje $L_{0,1,d}$, las cuales representan todas las posibles interpretaciones de las fórmulas booleanas en CNF y genera cadenas del lenguaje $L_{FULL-SAT}$, tales que la fórmula booleana asociada a estas cadenas es satisfacible.

El problema de la palabra para todo formalismo que genere el lenguaje $L_{0,1,d}$ y sea cerrado bajo transducción finita, es NP-Duro, teniendo en cuenta la conjetura de que cualquier formalismo que genere el lenguaje $L_{0,1,d}$, tiene tamaño $O(1)$ en su representación.

En el capítulo 4, se presentó una RCG que reconoce el lenguaje $L_{0,1,d}$ y se argumentó por qué no es posible usar esta gramática para construir L_{S-SAT} mediante transducción finita, ya que las RCG no son cerradas bajo transducción finita.

Se construyó una RCG que reconoce el lenguaje L_{S-SAT}, lo que permitió demostrar que no es necesario construir L_{S-SAT} mediante transducción finita. La gramática que se construyó tiene el problema de la palabra no polinomial, y constituye un ejemplo de una RCG donde el algoritmo de reconocimiento es no polinomial. Además al obtener una RCG que reconoce L_{S-SAT}, se demostró que las RCG cubren todos los problemas de la clase NP, ya que las RCG cubren todos los problemas en P [4] y existe una reducción polinomial del SAT a todo problema en NP [8].

Las estrategias presentadas constituyen una vía diferente para resolver el SAT, y aunque el problema de la palabra para el formalismo que se construyó es no polinomial, este acercamiento puede contribuir a nuevas líneas de investigación para la búsqueda de algoritmos eficientes que permitan resolver el SAT.

Recomendaciones

A partir del trabajo realizado se proponen como temas para investigaciones futuras los siguientes:

- Buscar un formalismo que sea capaz de generar el lenguaje $L_{0,1,d}$, el cual representa todas las interpretaciones de las fórmulas booleanas en CNF, que sea cerrado bajo transducción finita, y luego analizar el problema de la palabra para el formalismo que se obtiene después de aplicarle el transductor T_{SAT}.
- Demostrar que cualquier formalismo que genere $L_{0,1,d}$ tiene un tamaño $O(1)$ en su representación.
- Analizar qué tipo de formalismo se obtiene al aplicarle el transductor T_{SAT} a la RCG que reconoce el $L_{0,1,d}$.
- Analizar qué propiedades limitan que las RCG no sean cerradas bajo transducción finita, construir un formalismo basado en las RCG que sea cerrado bajo transducción finita y comprobar que este formalismo sea capaz de describir el lenguaje $L_{0,1,d}$.
- Construir una RCG que reconozca fórmulas booleanas satisfacibles, donde cada cláusula tiene a lo sumo dos literales (2-SAT), que tenga el problema de la palabra polinomial.

Referencias

[1] Alina Fernández Arias. «El problema de la satisfacibilidad booleana libre del contexto». En: (2007) (vid. págs. 5, 13, 14).

[2] Pierre Boullier. *A Cubic Time Extension of Context-Free Grammars*. Research Report RR-3611. INRIA, 1999. URL: `https://inria.hal.science/inria-00073067` (vid. pág. 21).

[3] Pierre Boullier. «Counting with range concatenation grammars». En: *Theor. Comput. Sci.* 293 (feb. de 2003), págs. 391-416. DOI: `10.1016/S0304-3975(01)00353-X` (vid. pág. 51).

[4] Pierre Boullier. *Proposal for a Natural Language Processing Syntactic Backbone*. Research Report RR-3342. INRIA, 1998. URL: `https://inria.hal.science/inria-00073347` (vid. págs. 14, 15, 21, 38, 50, 54).

[5] Cristian Calude, Kai Salomaa y Tania Roblot. «Finite-State Complexity and the Size of Transducers». En: *Electronic Proceedings in Theoretical Computer Science* 31 (ago. de 2010), págs. 38-47. ISSN: 2075-2180. DOI: `10.4204/eptcs.31.6`. URL: `http://dx.doi.org/10.4204/EPTCS.31.6` (vid. pág. 8).

[6] William F. Dowling y Jean H. Gallier. «Linear-time algorithms for testing the satisfiability of propositional horn formulae». En: *The Journal of Logic Programming* 1.3 (1984), págs. 267-284. ISSN: 0743-1066. DOI: `https://doi.org/10.1016/0743-1066(84)90014-1`. URL: `https://www.sciencedirect.com/science/article/pii/0743106684900141` (vid. pág. 13).

[7] Steven Halim y Felix Halim. *Competitive Programming 3: The New Lower Bound of Programming Contests*. 2-SAT Problem. Lulu.com, 2013, págs. 336-337 (vid. págs. 12, 13).

[8] John E. Hopcroft, Rajeev Motwani y Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd. Addison-Wesley, 2006. ISBN: 9780321455369 (vid. págs. 4-7, 10-12, 27, 54).

[9] Manuel Aguilera López. «Problema de la Satisfacibilidad Booleana de Concatenación de Rango Simple». En: (2016) (vid. págs. 5, 13, 14, 19).

[10] Leslie G. Valiant. «The Complexity of Enumeration and Reliability Problems». En: *SIAM Journal on Computing* 8.3 (1979), págs. 410-421. DOI: `10.1137/0208032`. eprint: `https://doi.org/10.1137/0208032`. URL: `https://doi.org/10.1137/0208032` (vid. pág. 13).