Funciones de Censo de Lenguajes Libres del Contexto

Luis Eduardo Zambrano Fernández

Universidad Industrial de Santander
Facultad de Ciencias
Escuela de Matemáticas
Bucaramanga
2011

Funciones de Censo de Lenguajes Libres del Contexto

Luis Eduardo Zambrano Fernández

Trabajo de grado para optar al título de
Magíster en Matemáticas

Director
Dr. Rer. Nat. Juan Andrés Montoya Argüello

Universidad Industrial de Santander
Facultad de Ciencias
Escuela de Matemáticas
Bucaramanga
2011

A Eduardo y Karina

AGRADECIMIENTOS

Muy especialmente a Juan Andrés Montoya Argüello, por su orientación en todo este exitoso proceso, por los ratos compartidos, por su saber hecho vida y por inspirarme a vivir para el saber.

A nuestros profesores, Edilberto José Reyes Gonzalez, Sonia Marleni Sabogal Pedraza, Julio Cesar Carrillo Escobar, Rafael Fernando Isaacs Giraldo, Javier Enrique Camargo García, Elder Jesús Villamizar Roa y José Bernardo Mayorga Rodríguez, por sus aportes en nuestra formación académica y personal.

Índice general

RESUMEN

TÍTULO: Funciones de Censo de Lenguajes Libres del Contexto[*].

AUTOR: Luis Eduardo Zambrano Fernández[**].

Palabras claves: Problemas de conteo delgado, método de Schutzenberger-Bertoni, poliominoes, caminos que se auto-evitan, complejidad computacional.

DESCRIPCIÓN:

En este trabajo, estudiamos algunos de los resultados más relevantes acerca de las funciones de censo de los lenguajes libres del contexto y usamos estos resultados para resolver dos problemas de conteo delgado (tally counting problems) cuando son restringidos a grillas rectangulares de altura fija. Estos problemas surgen de la tomografía discreta y la mecánica estadística. Nuestras soluciones se basan en el método de Schützenberger-Bertoni, una importante técnica de conteo que permite reducir varios problemas de conteo al cálculo de las funciones de censo de ciertos lenguajes formales.

La complejidad de calcular la función de censo de lenguajes libres del contexto está estrechamente relacionada con la ambigüedad del lenguaje. Para lenguajes regulares, existen algoritmos de tiempo lineal que permiten calcularla, para lenguajes libres del contexto no ambiguos, existen algoritmos en paralelo que permiten calcularla en tiempo polilogarítmico, sin embargo, para lenguajes libres del contexto ambiguos no se conocen algoritmos determinísticos de tiempo polinomial que permitan calcular su función de censo, solo se conocen esquemas de aproximación aleatorios completamente polinomiales que permiten aproximarla.

Nuestro principal aporte consiste en reducir el problema de conteo de poliominoes en grillas rectangulares de altura fija al cálculo de la función de censo de un lenguaje libre del contexto determinístico y reducir el problema de conteo de caminos que se auto-evitan en grillas rectangulares de altura fija al cálculo de la función de censo de un lenguaje regular.

[*] Trabajo de Grado.
[**] Facultad de Ciencias. Escuela de Matemáticas. Director: Prof. Dr. Juan Andrés Montoya Argüello.

ABSTRACT

TITLE: Census Functions of Context Free Languages.[*]

Author: Luis Eduardo Zambrano Fernández.[**]

Keywords: Tally counting problems, Schutzenberger-Bertoni Method, polyominoes, self-avoiding walks, computational complexity.

DESCRIPTION:

In this paper, we study some of the most relevant results on the census functions of context free languages and use these results to solve two tally counting problems when they are restricted to rectangular grids of fixed height. These problems arise from discrete tomography and statistical mechanics. Our solutions are based on the Schützenberger-Bertoni Method, a powerful counting technique that reduces multiple counting problems to the calculation of functions Census of certain formal languages.

The complexity of calculating the census function of context-free languages is closely related to the ambiguity of language. For regular languages, there are linear time algorithms to calculate, if context-free languages is not ambiguous, there are parallel algorithms that calculate in polylogarithm time, however, for ambiguous context-free languages are not known deterministic polynomial time algorithms that can calculate their function census, only known fully polynomial time randomized approximation schemes to approximate.

Our main contribution in this paper is to reduce the problem of counting polyominoes on rectangular grids of fixed height to the calculation the census function of a deterministic context-free language, and reduce the problem of counting self-avoiding walks on rectangular grids of fixed height to the calculation the census function of a regular language.

[*] Paper Work

[**] Faculty of Sciences. School of Mathematics. Director: Prof.Dr. Juan Andrés Montoya Argüello.

Capítulo 1

Introducción

En un lenguaje formal, la función que a cada entero no negativo n le asigna el cardinal del conjunto de las palabras de longitud n, es llamada la *función de censo* (*función de crecimiento* o *función de conteo*) del lenguaje. Las funciones de censo de los lenguajes libres del contexto han sido ampliamente investigadas en la literatura por razones que van más allá de la belleza matemática inherente a su estudio. Desde el punto de vista de la complejidad computacional, nuestro estudio ha permitido acercarnos a la solución de importantes problemas de conteo que tienen interés particular en mecánica estadística y tomografía discreta e interés general en matemáticas discretas, combinatoria y la teoría de autómatas y lenguajes formales.

La clasificación de problemas de conteo en clases de complejidad fue inicialmente investigada por Valiant en [29]; en este trabajo el autor muestra que computar la permanente de una matriz (o equivalentemente, contar el número de matchings perfectos en grafos bipartitos) es un problema #**P**-completo; intuitivamente, una prueba de que un cierto problema de conteo es #**P**-completo es una prueba fuerte de que no existen algoritmos eficientes que lo resuelvan. Posteriormente, Jerrum, Valiant y Vazirani [17] estudiaron la aproximabilidad de los problemas de conteo, logrando exhibir una útil e importante relación entre la generación aleatoria uniforme y el conteo aproximado. Más tarde, Flajolet, Van Cutsen y Zimmerman [11] lograron diseñar un algoritmo de generación uniforme para lenguajes no ambiguos; este algoritmo puede ser usado, explotando la conexión exhibida por Jerrum, Valiant y Vazirani, para dis-

eñar algoritmos de conteo aproximado para las funciones de censo de los lenguajes libres del contexto no ambiguos (las cuales pueden ser calculadas eficientemente de manera exacta [4]) y más interesante aún, para diseñar algoritmos probabilísticos de aproximación para las funciones de censo de los lenguajes libres de contexto de ambigüedad polinomial (para las cuales, no se conocen algoritmos eficientes que las calculen de manera exacta). Nosotros estudiaremos en profundidad esta importante consecuencia del trabajo de Flajolet.

Nuestro estudio de las funciones de censo inicia con un importante teorema de "*gap*" (Teorema 2.4), el cual expresa que no existen funciones de censo de crecimiento estrictamente superpolinomial y estrictamente subexponencial. Posteriormente estudiamos las funciones de censo de los lenguajes libres del contexto no ambiguos, es decir, lenguajes descritos por una gramática libre del contexto donde toda palabra en el lenguaje admite un único árbol de derivación. Mostramos que el cálculo de las funciones de censo de tales lenguajes resulta eficiente. No obstante, los lenguajes libres del contexto ambiguos presentan funciones de censo de difícil evaluación, de hecho, existen lenguajes libres del contexto de ambigüedad dos cuyas funciones de censo son $\#\mathbf{P}_1$-completas. Este hecho sugiere que, no todo lenguaje libre del contexto ambiguo admite un algoritmo eficiente que permita evaluar su función de censo; de modo que, el estudio de las funciones de censo de tales lenguajes ha sido abordado desde el punto de vista de la existencia de algoritmos probabilísticos que permiten aproximarla. Siguiendo la relación mostrada por Jerrum, Valiant y Vazirani entre la generación aleatoria uniforme y el conteo aproximado, introducimos algoritmos de generación uniforme para lenguajes libres del contexto no ambiguos y desde éstos, con cierto trabajo, se construyen algoritmos para generación aleatoria uniforme para lenguajes libres del contexto de ambigüedad finita. Finalmente, presentamos el método de Schützenberger-Bertoni, una técnica de conteo que nos permite reducir algunos problemas de conteo al cálculo de las funciones de censo de ciertos lenguajes formales. El Capítulo 4 exhibe reducciones de problemas de conteo de la tomografía discreta y mecánica estadística al cálculo de las funciones de censo de lenguajes libres del contexto.

Capítulo 2

Lenguajes libres del contexto y funciones de censo

En este capítulo introducimos algunas definiciones y nociones preliminares; asumiremos que el lector está familiarizado con los rudimentos de la teoría de autómatas y lenguajes formales, en particular, con los conceptos de *lenguaje libre del contexto (CFL), gramática libre del contexto (CFG), autómata de pila (PDA) y máquina de Turing (TM);* así, más que hacer un recuento de algunos conceptos básicos, este capítulo tiene por objeto establecer la notación que usaremos en todo el documento. Para una completa revisión sobre la teoría de autómatas y lenguajes formales, el lector interesado puede consultar las excelentes referencias [15] y [27].

En primer lugar, definiremos la noción de función de censo de un lenguaje formal. Al final de esta sección estudiaremos un interesante teorema relacionado con las funciones de censo de lenguajes libres del contexto, este teorema establece que las funciones de censo de los lenguajes libres del contexto tienen, exclusivamente, crecimiento polinomial o estrictamente exponencial, esto es, no existen lenguajes libres del contexto cuya función de censo sea de crecimiento intermedio. Nos referiremos a este teorema con el término *"Teorema del Gap"*.

2.1. Funciones de Censo

Definición 2.1 *Si $L \subseteq \Sigma^*$ es un lenguaje formal sobre un alfabeto Σ, la* función de censo *del lenguaje L, es la función $f_{_L} : \mathbb{N} \to \mathbb{N}$ definida como*

$$\forall n \in \mathbb{N},\ f_{_L}(n) = |L \cap \Sigma^n|$$

Definición 2.2 *Si L es un lenguaje formal con función de censo f_L diremos que:*

1. *L es de* crecimiento polinomial *si y solo si existe $n_0 \in \mathbb{N}$ y un polinomio $p(x) \in \mathbb{Q}[x]$ tal que $f_{_L}(n) \leq p(n)$ para todo $n \geq n_0$. Los lenguajes de crecimiento polinomial son frecuentemente llamados* dispersos (sparse) *o* p-escasos (poly-slender).

2. *L es de* crecimiento estrictamente exponencial *si y solo si existe un número real $r > 1$ tal que $f_{_L}(n) \geq r^n$ para un número infinito de valores de n. Los lenguajes de crecimiento exponencial son también llamados* densos.

3. *L es de* crecimiento intermedio *si y solo si $f_{_L}(n)$ es subexponencial y super-polinomial, es decir, $f_{_L}$ no está acotada superiormente por un polinomio y $\limsup \frac{f_L(n)}{r^n} = 0$ para todo $r > 1$.*

Usaremos el símbolo ϵ para denotar la palabra vacía sobre cualquier alfabeto Σ, además, usaremos w^* en lugar de $\{w\}^*$ para denotar la cerradura de Kleene del lenguaje que solo contiene la palabra w.

Definición 2.3 *Un lenguaje $L \subseteq \Sigma^*$, con $|\Sigma| \geq 2$, es* acotado *si existen palabras w_1,w_2, ... ,$w_r \in \Sigma^*$, no vacías y tales que $L \subseteq w_1^* w_2^* \cdots w_r^*$. Las palabras w_1,w_2, ... ,w_r son llamadas* palabras correspondientes *del lenguaje L.*

El siguiente lema caracteriza el crecimiento de las funciones de censo de los lenguajes acotados.

Lema 2.1 *Todo lenguaje acotado es de crecimiento polinomial.*

Prueba. *Suponga que L es un lenguaje acotado con palabras correspondientes $w_1, w_2, \ldots, w_r$, mostraremos por inducción sobre r que para todo $n \in \mathbb{N}$, $f_{_L}(n) \leq (n+1)^{r-1}$. Si $r = 1$, $L \subseteq w_1^*$, por lo tanto, para cada $n \in \mathbb{N}$, L tiene a lo más una palabra de longitud n, i.e., $f_{_L}(n) \leq 1$. Supongamos que la afirmación es cierta para $r-1$, i.e., $f_{_L}(n) \leq (n+1)^{r-2}$ se verifica para todo $n \in \mathbb{N}$. Por otra parte, si $w \in L \subseteq w_1^* w_2^* \cdots w_r^*$ y $|w| = n$, entonces $w = uv$ con $u \in w_1^* w_2^* \cdots w_{r-1}^*$, y $v \in w_r^*$, note que tanto u como v pueden ser vacías, por lo tanto, tal descomposición se puede hacer a lo más en $(n+1)$ posiciones. Como $u \in w_1^* w_2^* \cdots w_{r-1}^*$ y $|w| = n$, entonces $|u| \leq n$ y por hipótesis de inducción $f_{_L}(n) \leq (n+1)^{r-2}(n+1) = (n+1)^{r-1}$.* ■

El recíproco del Lema 2.1 no es cierto en general para todo lenguaje formal, *Raz* [24] muestra que el lenguaje

$$L = \{w\#1^i : i > 0 \ \& \ w \text{ es la representación binaria de } i\}$$

es un lenguaje dependiente del contexto de crecimiento polinomial que no es acotado. Sin embargo, para lenguajes libres del contexto, crecimiento polinomial implica acotación [6].

Definición 2.4 *Sean $u, v \in \Sigma^*$, diremos que u y v conmutan si $uv = vu$. Un subconjunto $C \subseteq \Sigma^*$ se dice que es conmutativo si $\forall u, v \in C,\ uv = vu$.*

Lema 2.2 *([13, Lemma 5.1]) Sean $u, v \in \Sigma^*$, las siguientes afirmaciones son equivalentes:*

- *u y v conmutan.*
- *$u^q = v^p$, para ciertos $p, q \geq 1$.*
- *$u = w^r$ y $v = w^s$, para alguna palabra $w \in \Sigma^*$, y para $r, s \geq 1$.*

Denotaremos una gramática libre del contexto G con la cuádrupla (Σ, V, P, S), donde Σ representa el conjunto de *símbolos terminales* (o el alfabeto de la gramática), V el conjunto de *variables*, P el conjunto de *reglas de producción* y S la *variable inicial* de la gramática. Usaremos el símbolo $L(G)$ para denotar el lenguaje generado por G.

Definición 2.5 *Sean $G = (\Sigma, V, P, S)$ una gramática libre del contexto, $A \in V$ y $\alpha \in (\Sigma \cup V)^*$. Una G-derivación de α desde A es una secuencia $\alpha_1, ..., \alpha_n$ de elementos de $(\Sigma \cup V)^*$ tal que:*

1. $\alpha_1 = A$.
2. $\alpha_n = \alpha$.
3. *Para todo $i \leq n-1$ se tiene que la expresión α_{i+1} puede ser obtenida a partir de α_i aplicando alguna de las reglas de producción en P a la variable que ocurre más a la izquierda en α_i.*

Dado $A \in V$ y dada $\alpha \in (\Sigma \cup V)^*$ usaremos la notación $A \overset{*}{\Rightarrow} \alpha$ para indicar que existe una G-derivación de α desde A.

Notación 2.1 *Sea $G = (\Sigma, V, P, S)$ una gramática libre del contexto sin variables inútiles*[1] *y sea $A \in V$ una variable de la gramática, definimos los conjuntos*

$$I_A(G) = \{u \in \Sigma^* : A \overset{*}{\Rightarrow} uAv, \text{ para algún } v \in \Sigma^*\} \tag{2.1}$$

$$D_A(G) = \{v \in \Sigma^* : A \overset{*}{\Rightarrow} uAv, \text{ para algún } u \in \Sigma^*\} \tag{2.2}$$

Teorema 2.1 *([13, Theorem 5.1]) Sea $G = (\Sigma, V, P, S)$ una gramática libre del contexto. Una condición necesaria y suficiente para que el lenguaje generado $L(G) \neq \varnothing$ sea acotado es que $I_A(G)$ y $D_A(G)$ sean conmutativos para cada variable $A \in V$.*

Usando el teorema anterior, Raz [24] prueba la siguiente proposición.

Proposición 2.1 *([24, Proposition 2]) Sea $G = (\Sigma, V, P, S)$ una gramática libre del contexto. Si $L(G)$ es de crecimiento polinomial, entonces $L(G)$ es acotado.*

Como consecuencia inmediata del Lema 2.1 y la Proposición 2.1 tenemos el siguiente corolario.

[1] Toda variable de la gramática participa en la derivación de al menos una palabra.

Corolario 2.1 *Un lenguaje libre del contexto es acotado si y solo si es de crecimiento polinomial.*

Ginsburg y Spanier prueban los dos siguientes teoremas.

Teorema 2.2 *[13, Theorem 5.2] Sea G una gramática libre del contexto, entonces:*

1. *Es decidible si el lenguaje $L(G)$ es acotado.*
2. *Si $L(G)$ es acotado, entonces es posible encontrar (de manera efectiva) palabras $w_1, w_2, \ldots, w_r \in \Sigma^*$ tales que $L \subseteq w_1^* w_2^* \cdots w_r^*$.*

Teorema 2.3 *[13, Theorem 6.3] Si L_1 y L_2 son lenguajes libres del contexto y uno de éstos dos lenguajes es acotado, entonces se puede determinar si*

1. *$L_1 \subseteq L_2$, o si*
2. *$L_2 \subseteq L_1$*

Corolario 2.2 *Si L_1 y L_2 son lenguajes libres del contexto y uno de éstos es acotado, entonces se puede determinar si $L_1 = L_2$.*

Concluiremos esta sección con el *Teorema del Gap* para funciones de censo de lenguajes libres del contexto.

Notación 2.2 *Si W es una palabra arbitraria sobre el alfabeto binario $\{0, 1\}$, y u, v son palabras sobre cualquier alfabeto Σ, entonces $W(u, v)$ denotará la palabra que resulta de sustituir en W cada 0 por u y cada 1 por v.*

Teorema 2.4 *Sean, $G = (\Sigma, V, P, S)$ una gramática libre del contexto sin variables inútiles, $L(G)$ el lenguaje generado por G y $f_{L(G)}$ la función de censo del lenguaje $L(G)$. Una y solo una de las siguientes afirmaciones se verifica*

1. *Existe un número $r > 1$ y un entero n_0 tal que $f_{L(G)}(n) \geq r^n$ para todo $n \geq n_0$, o bien*
2. *$L(G)$ es un lenguaje acotado*

Prueba. En razón al Teorema 2.1, es suficiente mostrar que si alguno de los conjuntos $I_A(G)$ o $D_A(G)$ no es conmutativo para alguna variable $A \in V$, entonces $L_A = \{w \in \Sigma^* : A \overset{*}{\Rightarrow} w\}$, el lenguaje formado por todas las palabras derivadas desde A, tiene una función de censo que es acotada inferiormente por una función exponencial. En efecto, puesto que G no tiene variables inútiles, A participa en la derivación de al menos una palabra en $L(G)$, así, si la función de censo del lenguaje L_A está acotada inferiormente por una función exponencial entonces la función de censo del lenguaje $L(G)$ también lo está.

Sin pérdida de generalidad, supongamos en particular que $I_A(G)$ no es conmutativo y sean $u_1, u_2 \in I_A(G)$ palabras que no conmutan. Entonces, existen palabras $v_1, v_2, w_1, w_2 \in \Sigma^*$ y $w_0 \in L_A$ tales que $A \overset{*}{\to} w_0$, $S \overset{*}{\to} w_1 A w_2$, $A \overset{*}{\to} u_1 A v_1$ y $A \overset{*}{\to} u_2 A v_2$ con $u_1 u_2 \neq u_2 u_1$. Sean además e, d enteros positivos tales que $|u_1^d| = |u_2^e|$ (e.g. $e = |u_1|$ y $d = |u_2|$) y considere la palabra en $L(G)$ producida al aplicar las reglas $S \overset{*}{\to} w_1 A w_2$ (una vez), $A \overset{*}{\to} u_1 A v_1$ (d veces), $A \overset{*}{\to} u_2 A v_2$ (e veces) y $A \overset{*}{\to} w_0$ (una vez); así, obtenemos $w_1 u_1^d u_2^e w_0 v_2^e v_1^d w_2 \in L(G)$; denotaremos esta palabra como $W(u_1^d, u_2^e) w_0 W^R(v_1^d, v_2^e)$, donde W^R denota el reverso de W. Note que si dos palabras distintas W y W' de la misma longitud k producen la misma palabra en $L(G)$, entonces, o $W(u_1^d, u_2^e)$ es prefijo de $W'(u_1^d, u_2^e)$, o $W'(u_1^d, u_2^e)$ es prefijo de $W(u_1^d, u_2^e)$; en nuestro caso, como $|u_1^d| = |u_2^e|$ entonces $W(u_1^d, u_2^e) = W'(u_1^d, u_2^e)$. Como W y W' son distintos, se debe tener que $u_1^d = u_2^e$ y de acuerdo al Lema 2.2, u_1 y u_2 conmutan. Puesto que por hipótesis u_1 y u_2 no conmutan, se concluye que las 2^k palabras W de longitud k inducen 2^k palabras distintas $W(u_1^d, u_2^e) w_0 W^R(v_1^d, v_2^e)$ en L_A. Si W es una palabra de tamaño k, y hacemos $l = \text{máx}\{d, e\}$ y m un entero positivo mayor o igual a cada uno de los tamaños $|u_1|, |u_2|, |v_1|, |v_2|$ y $|w_0|$, entonces $|W(u_1^d, u_2^e) w_0 W^R(v_1^d, v_2^e)| \leq (2k+1)ml$, así, la función de censo del lenguaje L_A, que denotaremos f_{L_A}, satisface $f_{L_A}((2k+1)ml) \geq 2^k$ si $k \geq 1$. Supongamos $n \geq 6ml$; dividiendo n por ml obtenemos $n = (2k+1)ml + r$ para algún $k \geq 1$ y $0 \leq r \leq 2ml$, por lo tanto $f_{L_A}(n) \geq 2^k$. Note que $n \leq (2k+1)ml + 2ml = 2kml + 3ml$, de aquí que $k \geq \frac{n-3ml}{2ml}$; como $n \geq 6ml$, se tiene $2n \geq 6ml + n$, luego $\frac{2n-6ml}{4ml} \geq \frac{n}{4ml}$, y de aquí, $\frac{n}{4ml} \leq \frac{n-3ml}{2ml} \leq k$. Esto implica que $f_{L_A}(n) \geq 2^k \geq 2^{\frac{n}{4ml}} = \left(2^{\frac{1}{4ml}}\right)^n$. Haciendo $r = 2^{\frac{1}{4ml}} > 1$, se tiene $f_{L_A}(n) \geq r^n$ siempre que $n \geq 6ml$. ■

Corolario 2.3 *No existen lenguajes libres del contexto de crecimiento intermedio.*

2.2. Ambigüedad

La problemática de la ambigüedad es de suma importancia en la teoría de los lenguajes (y gramáticas) libres del contexto. Recuerde ([15]) que las gramáticas libres del contexto son utilizadas para definir la gramática (sintaxis) de los lenguajes de programación, los cuales deben ser completamente no ambiguos. Por otro lado, como se hará patente en esta tesis, los lenguajes libres del contexto no ambiguos son *bien comportados*, en tanto que sus funciones de censo pueden ser calculadas eficientemente.

Definición 2.6 *Dadas* $G = (\Sigma, V, P, S)$, $A \in V$ *y* $\alpha \in (\Sigma \cup V)^*$ *usaremos el símbolo* $\#_A(\alpha)$ *para denotar el número de* G*-derivaciones de* α *desde* A.

Definición 2.7 *Dado* L *un lenguaje libre del contexto diremos que*

1. L *es un lenguaje* no ambiguo *si y solo si existe una gramática libre del contexto* G, *tal que* $L = L(G)$ *y para cada* $w \in L$ *se tiene que* $\#_S(w) = 1$.

2. L *es un lenguaje de* ambigüedad finita *si y solo si existe una gramática libre del contexto* G, *y existe una constante* K *tales que* $L = L(G)$ *y para cada* $w \in L$ *se tiene que* $\#_S(w) \leq K$.

3. L *es un lenguaje de* ambigüedad polinomial *si y solo si existe una gramática libre del contexto* G, *y existe un polinomio* $p(X)$ *tales que* $L = L(G)$ *y para cada* $w \in L$ *se tiene que* $\#_S(w) \leq p(|w|)$.

4. L *es un lenguaje de* ambigüedad estrictamente exponencial *si y solo si existe una gramática libre del contexto* G, *y existe* $r > 1$ *tal que* $L = L(G)$ *y para infinitos* w *en* L *se tiene que* $\#_S(w) \geq r^{|w|}$.

Es interesante anotar que existe un *teorema de Gap* asociado a la noción de ambigüedad y similar al Teorema 2.4 de Bridson y Gillman [6].

Teorema 2.5 *Dado L un lenguaje libre del contexto, o L es de ambigüedad polinomial o L es de ambigüedad estrictamente exponencial.*

El lector interesado en la prueba del Teorema 2.5 puede consultar la referencia [33].

Capítulo 3

Complejidad computacional de las funciones de censo de los lenguajes libres del contexto

En este capítulo estudiaremos la complejidad temporal de las funciones de censo de los lenguajes libres del contexto. Iniciaremos nuestro estudio con las definiciones de las clases $\#\mathbf{P}$, $\#\mathbf{P}_1$ y con la definición de $\#\mathbf{P}_1$-*completez* introducida por Leslie Valiant en [29]; centraremos nuestra atención en una clase particular de problemas de conteo, los así llamados *problemas de conteo delgados (tally counting problems)*; esta clase contiene una gran variedad de problema interesantes que ocurren en la práctica. En el Capítulo 4 estudiaremos algunos de estos problemas y presentaremos soluciones algorítmicas eficientes para tales problemas.

3.1. $\#\mathbf{P}$, $\#\mathbf{P}_1$ y problemas completos

Definición 3.1 *Una* máquina de Turing de conteo (CTM) *es una máquina de Turing $\mathcal{M}$, no determinística y provista con un dispositivo de salida auxiliar que (mágicamente), con input w, imprime en notación binaria sobre una cinta especial el número de computaciones aceptantes de $\mathcal{M}$ en el input w.*

Diremos que una CTM tiene *complejidad temporal (o tiempo de computo)* $f(n)$ si y solo si la computación más larga, inducida por cualquier input de longitud n, toma $f(n)$ pasos (cuando la CTM es considerada como una TM no determinística sin dispositivo auxiliar). Más aún, diremos que una función $f : \{0,1\}^* \longrightarrow \mathbb{N}$ es computable por una CTM, digamos $\mathcal{M}$, si para cada $x \in \{0,1\}^*$, $f(x)$ es el número de computaciones aceptantes de M en el input x.

Definición 3.2 $\#\mathbf{P}$ *es la clase de todas funciones que pueden ser calculadas por una CTM con tiempo de computo polinomial.*

Dada $\mathcal{M}$ una CTM, diremos que $\mathcal{M}$ es una CTM unaria si y solo si el alfabeto de entrada de $\mathcal{M}$ es unario.

Definición 3.3 $\#\mathbf{P}_1$ *es la clase de todas funciones que pueden ser calculadas por una CTM unaria cuyo tiempo de computo es polinomial.*

Denotaremos con **FP** a la clase de funciones que pueden ser calculadas por máquinas de Turing determinísticas de tiempo polinomial.

Definición 3.4 *Una* máquina de Turing con oráculo (TMO) *es una TM, digamos* $\mathcal{M}$*, dotada con una cinta de consulta, una cinta de respuesta, algunas cintas de trabajo y una función (la función oráculo de* $\mathcal{M}$*)* $h_M : \{1\}^* \longrightarrow \mathbb{N}$*. Para consultar el oráculo, la máquina* $\mathcal{M}$ *imprime una cadena unaria* 1^n *sobre la cinta de consulta, entra a un estado especial de consulta y retorna en una unidad de tiempo la representación binaria de* $h_M(1^n)$ *escribiéndola sobre la cinta de respuesta.*

Definición 3.5 *Dadas dos funciones* $f, g : \{1\}^* \longrightarrow \mathbb{N}$*, diremos que* f *es* #1-reducible *a* g *si y solo existe una TMO, digamos* $\mathcal{M}$*, con función oráculo* g *que puede calcular* f *en tiempo polinomial. Más aún, para cada input* 1^n*, la maquina* $\mathcal{M}$ *imprime la representación binaria de* $f(1^n)$ *en tiempo polinomial.*

Proposición 3.1 *Si* L *es un lenguaje en* P*, su función de censo pertenece a* $\#\mathbf{P}_1$*.*
Prueba. *Si* $\mathcal{M}$ *es una TM determinística que reconoce* L *en tiempo polinomial, entonces construimos una TM no determinística, digamos* $\mathcal{M}'$*, tal que, en el input*

1^n, *$\mathcal{M}'$ puede generar no determinísticamente cualquier cadena de longitud n, entonces, teniendo una cadena y de longitud n sobre su cinta de entrada, $\mathcal{M}'$ simula la computación de la máquina $\mathcal{M}$ en el input y. Es claro que $\mathcal{M}'$ trabaja en tiempo polinomial y el número de computaciones aceptantes de $\mathcal{M}'$ en el input 1^n es igual a* $f_L(1^n)$. ■

Definición 3.6 *Diremos que una función $f \in \#\mathbf{P}_1$ es* $\#\mathbf{P}_1$-completa *si y solo si toda función $g \in \#\mathbf{P}_1$ es $\#1$-reducible a f.*

Es posible probar que existen problemas completos para $\#\mathbf{P}_1$. Sea $\mathcal{M}$ una CTM unaria, y sea $acc_{\mathcal{M}}(n)$ la función definida por: dado $n \geq 1$ la cantidad $acc_{\mathcal{M}}(n)$ es igual a el número de computaciones aceptantes de la máquina $\mathcal{M}$ en el input n. Note que para toda CTM unaria $\mathcal{M}$ la función $acc_{\mathcal{M}}(n)$ es un problema de conteo delgado.

Teorema 3.1 *Existe una CTM unaria, digamos $\mathcal{M}$, tal que $acc_{\mathcal{M}}(n)$ es $\#\mathbf{P}_1$ completo.*

Para una prueba el lector puede consultar la referencia [30].

3.2. Problemas de conteo delgados

Un problema de conteo es una función $f : \Sigma^* \to \mathbb{N}$. Un problema de conteo $f : \Sigma^* \to \mathbb{N}$ es un *problema de conteo delgado* si y solo si el cardinal de Σ es igual a 1. Si $f : \Sigma^* \to \mathbb{N}$ es un problema de conteo delgado, asumiremos que Σ es igual a $\{1\}$. Existen muchos problemas de conteo delgados interesantes, considere por ejemplo los siguientes:

1. $f_{LS} : \Sigma^* \to \mathbb{N}$ es el problema de conteo delgado definido por

$$f_{LS}(1^n) = \#\text{ de cuadrados latinos de orden } n$$

2. $f_M : \Sigma^* \to \mathbb{N}$ es el problema de conteo delgado definido por

$$f_M(1^n) = \#\text{ de cuadrados mágicos de orden } n, \text{ cuyas sumas-lineales son iguales a } n$$

3. $f_G : \Sigma^* \to \mathbb{N}$ es el problema de conteo delgado definido por

$$f_G(1^n) = \#\text{ de grupos de orden } n$$

Nosotros estamos particularmente interesados en el siguiente problema de conteo delgado.

Problema 3.1 *(#SAW, the self-avoiding problem)*

Input: 1^n, *donde* $n \in \mathbb{N}$.

Output: *Calcular el número de caminos simples de cualquier longitud contenidos en la grilla cuadrada de orden* n.

Este problema es importante porque codifica la función de partición del modelo SAW (Self-avoiding walk model) de la termodinámica de polímeros [32]. No se conoce algoritmo eficiente que resuelva $\#SAW$ y se desconoce si $\#SAW$ es un problema duro (en algún sentido, no existen pruebas de dureza algorítmica para este problema).

Considere ahora el siguiente problema.

Problema 3.2 *(Problema de Welsh) Exhiba un algoritmo de tiempo polinomial que resuelva* $\#SAW$*, o pruebe que* $\#SAW$ *es* $\#\mathbf{P}_1$*-completo [32].*

Desafortunadamente, no resolveremos el problema de Welsh, pero en el Capítulo 4 consideraremos algunas restricciones de $\#SAW$ y probaremos algunos resultados concernientes a tales restricciones, estudiaremos la función de partición de una versión restringida del modelo $\#SAW$.

Observación 3.1 *Lo que es especial y sospechoso acerca de los problemas de conteo delgados es la codificación unaria de sus instancias. Considere por ejemplo el problema* $\#SAW$*. Dado* $n \geq 1$*, el espacio de búsqueda asociado a la instancia* 1^n *es* $([n] \times [n])^n$ *y su tamaño es igual a* $2^{\Omega(n \log(n))}$*; esto implica que el tiempo de cómputo del algoritmo de fuerza bruta para* $\#SAW$ *es igual a* $2^{\Omega(n \log(n))}$*, lo cual muestra que los problemas de conteo delgados no (necesariamente) resultan tratables debido*

a la codificación unaria de sus instancias. Ahora bien, si usamos codificación binaria para sus instancias, obtendríamos un problema tal que el algoritmo ingenuo de fuerza bruta que lo resuelve es de tiempo super-exponencial, un problema que además no podría ser resuelto en tiempo polinomial debido al tamaño de los outputs, que serán de tamaño exponencial respecto al tamaño del input respectivo. Así, la búsqueda de algoritmos de tiempo polinomial (o algoritmos de tiempo polilogarítmico) que resuelvan el problema #SAW tiene sentido si consideramos la codificación unaria usada en la definición 3.1. Es claro ahora que la codificación unaria empleada en la definición de los problemas de conteo delgados no es un truco ingenuo diseñado para hacer que problemas (posiblemente) intratables parezcan fáciles.

3.3. Funciones de censo de lenguajes libres del contexto no ambiguos

En esta sección estudiaremos la clase de funciones de censo de los lenguajes libre del contexto no ambiguos. Como todo lenguaje libre del contexto está en $\mathbf{P}$, las funciones de censo de estos lenguajes están en $\#\mathbf{P}_1$ por lo que dado $L \in \mathcal{CFL}$ la función de censo de L podría ser o $\#\mathbf{P}_1$-completa o computable en tiempo polinomial. Probaremos en esta sección que si L es un lenguaje libre del contexto no ambiguo, entonces su función de censo f_L, puede ser calculada en tiempo $O\left(\log^2(n)\right)$ usando un número polinomial de procesadores. La prueba del teorema se basa fuertemente en el teorema de Chomsky-Schützenberger (Teorema 3.2) y un teorema debido a Comtet [8].

3.3.1. Lenguajes algebraicos

Una serie de potencias sobre $\mathbb{N}$ centrada en cero (en adelante, una *serie de potencias*), es una expresión de la forma

$$F(z) = \sum_{n=0}^{\infty} a_n z^n \text{ con } a_n \in \mathbb{N}$$

Definición 3.7 *Diremos que una serie de potencias F es* algebraica *sobre el anillo $\mathbb{Q}[x]$ si existen polinomios $p_0(x), p_1(x), \ldots, p_d(x) \in \mathbb{Q}[x]$ tales que la ecuación*

$$\sum_{j=0}^{d} p_j \left(F(x)\right)^j = 0 \tag{3.1}$$

se verifica. Diremos que d es el grado *de F si d es el menor entero para el que la ecuación 3.1 se satisface.*

Definición 3.8 *Si L es un lenguaje libre del contexto con función de censo f_L, la función generatriz de L, es la serie de potencias*

$$F(z) = \sum_{n=0}^{\infty} f_{_L}(n) z^n$$

Puesto que para cada $n \in \mathbb{N}$ se tiene que $f_{_L}(n) \leq |\Sigma^n| = |\Sigma|^n$, donde Σ es el alfabeto de L, se tiene entonces que el radio de convergencia ρ de la función generatriz $F(z)$ satisface $\frac{1}{|\Sigma|} \leq \rho < 1$.

Definición 3.9 *Dado L un lenguaje libre del contexto diremos que L es algebraico si y solo si la función generatriz de L es algebraica.*

Teorema 3.2 *(Chomsky-Schützenberger [7]) Si L es un lenguaje libre del contexto no ambiguo, entonces L es algebraico.*

3.3.2. Circuitos booleanos

En esta subsección introducimos un primer modelo de computación en paralelo, el modelo de circuitos booleanos.

Un circuito booleano tiene una cantidad fija, digamos n, de variables de entrada que le permiten evaluar (únicamente) vectores booleanos n-dimensionales (inputs de tamaño n). Sin embargo, puesto que estamos interesados en procesar instancias de tamaño arbitrario, consideraremos *familias* de circuitos booleanos, esto es, un circuito distinto para cada posible tamaño del input.

Definición 3.10 *Un* circuito booleano *es un digrafo $C = (V, E)$, donde los nodos en $V = \{1, \ldots, n\}$ son llamados las* compuertas *de C. El digrafo C está dotado de una estructura especial (la sintaxis):*

1. *No existen ciclos en el grafo, así, podemos asumir que todas las aristas son de la forma (i, j) con $i < j$.*

2. *Todos los nodos en el grafo tienen grado de entrada igual a 0, 1 o 2.*

3. *Cada compuerta $i \in V$ tiene asociado un tipo de expresión booleana $s_i \in \{\mathbf{1}, \mathbf{0}, \vee, \wedge, \neg\} \cup \{x_1, x_2, \ldots\}$ donde las x_i son variables booleanas.*

4. *Si $s_i \in \{\mathbf{1}, \mathbf{0}\} \cup \{x_1, x_2, \ldots\}$ entonces el grado de entrada de la compuerta i es cero, esto es, i no tiene aristas entrantes. Las compuertas con grado de entrada cero son llamadas* compuertas de entrada *del circuito* C.

5. *Si $s_i = \neg$, entonces i tiene grado de entrada uno.*

6. *Si $s_i \in \{\vee, \wedge\}$, entonces i tiene grado de entrada dos.*

7. *Existe un nodo, que asumiremos es el nodo n, cuyo grado de salida es igual a cero, diremos que este nodo es la* compuerta de salida *del circuito C.*

En la práctica, se consideran circuitos booleanos con varias compuertas de salida, en este caso, el conjunto formado por las compuertas que no tienen aristas salientes es considerado la salida *del circuito. Para definir la semántica del circuito, considere $X(C) = \{s_i : s_i \in \{x_1, x_2, \ldots\}$ para alguna arista i en $C\}$, el conjunto de todas las variables booleanas involucradas en el circuito C. Diremos que una asignación de verdad T es* apropiada *para C si ésta es definida para todas las variables en $X(C)$. Dada una asignación apropiada T, el* valor de verdad *de la compuerta $i \in V$, denotado $T(i)$, es definido inductivamente (sobre i) como sigue: Si $s_i = \mathbf{1}$ entonces $T(i) = \mathbf{1}$; si $s_i = \mathbf{0}$ entonces $T(i) = \mathbf{0}$; si $s_i \in X(C)$, entonces $T(i) = T(s_i)$; si $s_i = \neg$ entonces existe una única compuerta $j < i$ tal que $(j, i) \in E$, por inducción, conocemos el valor de $T(j)$, así $T(i) = \mathbf{0}$ si $T(j) = \mathbf{1}$ y viceversa; si $s_i = \vee$ entonces existen dos aristas que entran en i, digamos (j, i) y (j', i), entonces $T(i) = \mathbf{1}$ si y solo*

si al menos uno de $T(j)$ *o* $T(j')$ *es* $\mathbf{1}$*; análogamente, si* $s_i = \wedge$ *entonces* $T(i) = \mathbf{1}$ *si y solo si ambos* $T(j)$ *y* $T(j')$ *son* $\mathbf{1}$*, como antes,* (j, i) *y* (j', i) *son las aristas entrantes a* i*. Finalmente, el* valor *del circuito* $T(C)$*, es* $T(n)$*, donde* n *es la compuerta de salida de* C*.*

No es difícil probar que para cada función booleana n-aria f, existe un circuito booleano C_f con n variables de entrada que puede calcular f [22, 4.3]. En consecuencia, podemos pensar que un circuito con n variables de entrada *acepta* ciertas palabras de longitud n en el conjunto $\{0,1\}^*$ y rechaza las otras. Una cadena $x = x_1 \cdots x_n \in \{0,1\}^n$ es interpretada como una asignación de verdad a las variables de entrada del circuito, sin embargo, esta correspondencia es buena solo para palabras de longitud n. Con el fin de relacionar circuitos con lenguajes arbitrarios sobre el alfabeto $\{0,1\}$, necesitaremos considerar un circuito para cada posible longitud de la cadena de entrada, esto es, *familias* de circuitos booleanos.

Definición 3.11 *El* tamaño *de un circuito es el número de compuertas en éste. La* profundidad *del circuito es el número de compuertas en el camino más largo del circuito. Una* familia *de circuitos es una sucesión* $\{C_n\}_{n\in\mathbb{N}}$ *de circuitos booleanos, donde para cada* $n \in \mathbb{N}$ *el circuito* C_n *tiene* n *variables de entrada. Diremos que un lenguaje* $L \subseteq \{0,1\}^*$ *pertenece a* $\mathbf{P/poly}$ *si y solo si existe una familia de circuitos* $\{C_n\}_{n\in\mathbb{N}}$ *y un polinomio* $p(X)$ *tales que*

1. *Dado* $n \geq 1$ *el tamaño de* C_n *es a lo más* $p(n)$*, y*
2. *Para cada* $x \in \{0,1\}^*$ *se tiene que* $x \in L$ *si y solo si la salida de* $C_{|x|}$ *es* $\mathbf{1}$ *cuando las variables de entrada del circuito* $C_{|x|}$ *toman la asignación de verdad inducida por* x*.*

Un interesante resultado acerca de los lenguajes con circuitos booleanos es el siguiente.

Proposición 3.2 *([22, Proposition 11.1])* $\mathbf{P} \subseteq \mathbf{P/poly}$*.*

El recíproco de la Proposición 3.2 no es cierto, de hecho, existen lenguajes no decidibles que tienen circuitos polinomiales, la construcción de los circuitos para tales

lenguajes requiere cantidades exageradas de recursos de cómputo. Este hecho nos lleva a cuestionarnos si el modelo de *familias de circuitos booleanos* es un modelo de computación realista. Para no admitir lenguajes no decidibles, es necesario considerar familias *uniformes* de circuitos booleanos.

Definición 3.12 *Una familia de circuitos booleanos se dice que es* uniforme *si existe una máquina de Turing, la cual, en el input* 1^n*, calcula una descripción de* C_n *usando espacio logarítmico en* n*. Diremos que un lenguaje* L *puede ser reconocido por* circuitos polinomiales uniformes *si existe una familia uniforme de circuitos polinomiales que decide* L*.*

Proposición 3.3 *Un lenguaje* L *tiene circuitos polinomiales uniformes si y solo si* $L \in \mathbf{P}$*.*

Puesto que nuestro objetivo en este capítulo es estudiar la complejidad computacional de las funciones de censo de los lenguajes libres del contexto y dado que las familias uniformes de circuitos booleanos son nuestro modelo de computación en paralelo, es necesario definir medidas de complejidad, como por ejemplo el *tiempo de computo paralelo.*

Definición 3.13 *Sea* $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ *una familia uniforme de circuitos booleanos y sean* $t(n)$ *y* $h(n)$ *funciones de valor entero. Diremos que el* tiempo en paralelo *de* $\mathcal{C}$ *es* $t(n)$ *si para todo* n *la profundidad de* C_n *es a lo más* $t(n)$*.*

Definiremos las clase de complejidad **NC** como la clase de las funciones (problemas) computables mediante familias uniformes de tamaño polinomial y tiempo paralelo (profundidad) polilogarítmico.

Definición 3.14 *Para cada entero* k*, la clase* $\mathbf{NC}^k$ *es el conjunto de problemas computables por familias uniformes de circuitos booleanos de profundidad* $O(\log^k n)$ *y tamaño* $n^{O(1)}$*. La clase* **NC** *es definida como*

$$\mathbf{NC} = \bigcup_{k=1}^{\infty} \mathbf{NC}^k$$

Puesto que la cantidad de trabajo involucrada en resolver cualquier problema en **NC** está (por definición) acotada por un polinomio, es claro que $\mathbf{NC} \subseteq \mathbf{P}$. Es un problema abierto de la complejidad computacional determinar si $\mathbf{NC} = \mathbf{P}$. Por otro lado, si C es un circuito con n compuertas de entrada, tamaño $h(n)$ y profundidad $t(n)$, y v es un vector booleano de dimensión n, es posible calcular $v(C)$ en $t(n)$ unidades de tiempo empleando $h(n)$ procesadores. La observación anterior implica que si $L \in \mathbf{NC}^k$ el problema L puede ser resuelto en tiempo $O\left(\log^k n\right)$ usando un número polinomial de procesadores (i.e. puede ser resuelto en tiempo polilogarítmico en paralelo). Dado $k \geq 1$ usaremos el símbolo $\mathbf{FNC}^k$ para denotar la versión funcional de $\mathbf{NC}^k$, esto es: dada $f : \{0,1\}^* \to \mathbb{N}$ diremos que f pertenece a $\mathbf{FNC}^k$ si y solo si la función f puede ser calculada (i.e. puede ser evaluada en cualquiera de los elementos de su dominio) mediante una familia uniforme de circuitos booleanos de tamaño polinomial y profundidad $O\left(\log^k n\right)$.

3.3.3. Un teorema de Comtet.

La solución al problema de calcular eficientemente las funciones de censo de los lenguajes libres del contexto no ambiguos se basa fuertemente en el teorema de Chomsky-Schützenberger (Teorema 3.2) y en el siguiente teorema debido a Comtet [8].

Teorema 3.3 *(Teorema de Comtet, [8]) Sea $F(z) = \sum_{n=0}^{\infty} c_n z^n$ una serie de potencias algebraica de grado d, implícitamente definida por una ecuación de la forma*

$$\sum_{j=0}^{d} q_j(z)(F(z))^j = 0 \tag{3.2}$$

con $q_j \in \mathbb{Z}[X]$ para toda $j \in \{0, \ldots, d\}$. Entonces, existe un entero $n_0 \geq 0$ y polinomios $p_0, ..., p_k \in \mathbb{Z}[X]$ tales que

1. $p_0(X) \neq 0$
2. $\deg(p_j) < d$ *para cada* $0 \leq j \leq k$

3. Para todo $n \geq n_0$ se tiene que

$$p_0(n)\, c_n + p_1(n)\, c_{n-1} + ... + p_k(n)\, c_{n-k} = 0 \tag{3.3}$$

.

Prueba. (Esquema de la prueba)

Sea $Y = F(X)$ una función (serie) que satisface la ecuación polinomial 3.2, es decir,

$$q_0(X) + q_1(X)Y + \cdots + q_d(X)Y^d = 0 \tag{3.4}$$

Diferenciando 3.4 y usando inducción podemos obtener una expresión racional para cada una de las funciones $Y^{(k)} = d^k Y/dX^k$, con $k \geq 0$. Puesto que Y es algebraica de grado d sobre $\mathbb{Q}[X] \subset \mathbb{C}[X]$, la funciones 1, $Y^{(0)} = Y$, $Y^{(1)}$, ..., Y^{d-1} son linealmente dependientes sobre $\mathbb{C}[X]$. Podemos escribir ecuaciones que expresen estas relaciones de dependencia, en estas ecuaciones podemos eliminar los denominadores, logrando que los coeficientes de cada $Y^{(k)}$ sean polinomios en la variable X, posteriormente podemos expandir cada $Y^{(k)}$ como una serie de potencias en X, y finalmente, si igualamos los correspondientes coeficientes de X^n a ambos lados de las ecuaciones que expresan las relaciones de dependencia, obtenemos el resultado deseado (el lector interesado en una prueba más detallada puede consultar la referencia [8]). ■

3.3.4. #UCFL⊆FNC^2

En esta subsección usaremos el teorema de Chomsky-Schützenberger (Teorema 3.2) y el teorema de Comtet (Teorema 3.3) para mostrar que la clase de las funciones de censo de los lenguajes libres del contexto no ambiguos, que denotamos #**UCFL**, pueden ser calculadas por familias de circuitos booleanos de profundidad $O(\log^2 n)$ y tamaño polinomial.

Iniciaremos nuestro estudio considerando el problema de calcular los coeficientes de Taylor de una serie de potencias algebraica.

Sea $F : \mathbb{C} \to \mathbb{C}$ una función que es analítica en una vecindad de cero, considere el siguiente problema.

Problema 3.3 *($CT(F)$ cálculo de los coeficientes de Taylor de F)*

Input: 1^n

Output: *Calcule el n-ésimo coeficiente de Taylor de la serie de potencias de F alrededor de cero e imprima el resultado en binario.*

Observación 3.2 *La computabilidad del problema $CT(F)$ dependerá de F y de la manera como se presente la función F que define el problema. Si F es algebraica de grado d y está implícitamente definida por una ecuación de la forma*

$$\sum_{j=0}^{d} q_j(z)\,(F(z))^j = 0$$

donde $q_j \in \mathbb{Z}[X]$ para todo $j \in \{0, ..., d\}$. Podemos entonces presentar la función F mediante la tupla $(q_0, ..., q_d)$. En este caso, es posible recurrir al Teorema de Comtet para mostrar que el problema $CT(F)$ puede ser resuelto de manera eficiente.

Diremos que un problema tiene *profundidad booleana* $O(t(n))$ y *tamaño* $O(h(n))$ si puede ser resuelto por una familia uniforme de circuitos booleanos de profundidad $O(t(n))$ y tamaño $O(h(n))$. En nuestra prueba necesitaremos emplear una larga lista de resultados clásicos relacionados con la computabilidad de ciertos problemas usando familias uniformes de circuitos booleanos.

Lema 3.1 *([25, Theorem 3.2]) El problema consistente en calcular el producto de dos enteros de n bits tiene profundidad booleana $O(\log n)$ y tamaño $O(n \log n \log\log n)$.*

Corolario 3.1 *El problema de calcular la suma y el producto de dos enteros de $O(n)$ bits yace en $\mathbf{NC}^1$.*

Lema 3.2 *([25, Theorem 3.3]) Si n es una potencia de 2, el producto de m enteros de n bits módulo 2^n+1 tiene profundidad booleana $O(\log m \log\log n + \log n)$ y tamaño $(mn)^{O(1)}$.*

Corolario 3.2 *El problema de calcular el producto de n enteros de n bits yace en* $\mathbf{NC}^2$.

Lema 3.3 *([25, Corollary 3.4]) El inverso multiplicativo de un número no nulo de n bits puede ser computado con precisión $o(2^{-n})$ por una familia de circuitos de profundidad $O(\log n \log\log n)$ y tamaño $n^{O(1)}$.*

Lema 3.4 *([25, Corollary 3.5]) Dados los enteros a, b de n bits, podemos calcular enteros q y r tales que $a = qb + r$, con $0 \leq r < q$ usando una familia uniforme de circuitos booleanos de profundidad $O(\log n \log\log n)$ y tamaño $n^{O(1)}$.*

Corolario 3.3 *El problema de calcular el cociente de dos enteros de n bits yace en* $\mathbf{NC}^2$.

Lema 3.5 *([4, Lemma 2.8]) Sean $B_1, B_2, \ldots, B_n$ matrices de tamaño $k \times k$ con entradas enteras de $O(\log n)$ bits cada una y suponga que todas las entradas de la matriz $A = \prod_{i=1}^{n} B_i$ son enteros no negativos. Entonces el problema de calcular la matriz A tiene profundidad booleana $O(\log n \log\log n)$ y tamaño polinomial.*

A continuación mostraremos que es posible calcular eficientemente los coeficientes de Taylor para series de potencias adecuadas.

Proposición 3.4 *Si $F(z)$ es una serie de potencias que satisface la hipótesis del Teorema de Comtet, entonces el Problema $CT(F)$ yace en* $\mathbf{NC}^2$.

Prueba. Supongamos $F(z) = \sum_{n=0}^{\infty} c_n z^n$ algebraica de grado d implícitamente definida por la tupla $(q_0, ..., q_d)$. Por el Teorema 3.3 existe $n_0 \geq 0$ y polinomios $p_0, ..., p_k \in \mathbb{Z}[X]$ tales que para cada $n \geq n_0$ se cumple que

$$p_0(n)\, c_n + p_1(n)\, c_{n-1} + ... + p_k(n)\, c_{n-k} = 0 \tag{3.5}$$

Como $p_0(X) \neq 0$, para cada $i \in \{1, \ldots, k\}$ podemos hacer $r_i = -\frac{p_i(n)}{p_0(n)}$; así, de la ecuación 3.5 se tiene

$$c_n = r_1(n)\, c_{n-1} + ... + r_k(n)\, c_{n-k} \tag{3.6}$$

siempre que $n \geq n_0$. Considere ahora la sucesión de vectores

$$y(n) = \begin{pmatrix} c_{n-1} \\ c_{n-2} \\ \vdots \\ c_{n-k} \end{pmatrix}, y(n+1) = \begin{pmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_{n-k+1} \end{pmatrix}, \ldots$$

y la matriz de tamaño $k \times k$

$$A(n) = \begin{pmatrix} r_1(n) & r_2(n) & \cdots & r_{k-1}(n) & r_k(n) \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

Note que para cada $n \geq n_0$ (usando la ecuación 3.6) es posible formar la relación de recurrencia $y(n+1) = A(n)y(n)$, $y(n+2) = A(n+1)y(n+1) = A(n+1)A(n)y(n)$, ..., $y(n+l) = \left(\prod_{j=1}^{l} A(n+(l-j))\right) y(n)$, ...

Note además que calcular c_n es calcular el primer elemento de $y(n+1)$. Como la relación se cumple para todo $n \geq n_0$, si definimos M_0 como la matriz $k \times k$ cuya primera columna es el vector $y(n_0)$ y tal que todas sus otras entradas son cero, es fácil ver que calcular c_n es equivalente a calcular la componente $(1,1)$ de la matriz

$$M(n) = \left(\prod_{j=n_0}^{n} A(j)\right) M_0$$

Por otro lado, como la primera fila de la matriz $A(n)$ son expresiones racionales con denominador $p_0(n)$, hacemos $B(n) = p_0(n)A(n)$, que es una matriz $k \times k$ cuyas componentes son polinomios con coeficientes enteros. Note que $A(n) = \frac{1}{p_0(n)}B(n)$,

por lo tanto

$$M(n) = \left(\frac{1}{\prod_{j=n_0}^{n} p_0(j)} \prod_{j=n_0}^{n} B(j) \right) M_0 = \frac{1}{\prod_{j=n_0}^{n} p_0(j)} \left(\left(\prod_{j=n_0}^{n} B(j) \right) M_0 \right)$$

Para calcular c_n, esto es, la componente $(1,1)$ de la matriz $M(n)$ usamos el algoritmo $\mathcal{N}$ definido a continuación:

Input $n \geq 0$

Output c_n

1. Calcula la representación binaria de n y calcula en paralelo las matrices $B(n_0)$, $B(n_0+1)$, ..., $B(n)$ y los enteros (de $O(\log(n))$ bits) $p_0(n_0)$, $p_0(n_0+1)$, ..., $p_0(n)$.
2. Calcula el vector $y(n_0)$ y la matriz M_0.
3. Calcula el entero $b = \prod_{j=n_0}^{n} p_0(j)$
4. Calcula la matriz $M'(n) = \left(\prod_{j=n_0}^{n} B(j) \right) M_0$ y determina $M'_{11}(n)$ (la componente $(1,1)$ de esta matriz).
5. Calcula el entero $\frac{M'_{11}(n)}{b} = c_n$

El cálculo del Paso (1) yace en $\mathbf{NC}^1$ puesto que la suma y producto de dos enteros de $O(n)$-bits yace en $\mathbf{NC}^1$ (ver Corolario 3.1). Además, puesto que n_0 es fijo y no depende del input, el vector $y(n_0)$ y la matriz M_0 del Paso (2) pueden ser calculados en tiempo constante. Por otra parte, en razón al Corolario 3.2, calcular el entero b (de $O(n \log n)$ bits) del Paso (3) yace en $\mathbf{NC}^2$. Más aún, el problema de calcular $M'(n)$ en el Paso (4) yace en $\mathbf{NC}^2$ (ver Lema 3.5). Finalmente, debido al Corolario 3.3 el problema de calcular el cociente del Paso (5) yace en $\mathbf{NC}^2$. De todo esto, el problema de calcular los coeficientes c_n de la serie de potencias $F(z)$ pertenece a la clase $\mathbf{NC}^2$. ■

Corolario 3.4 *Si L es un lenguaje libre del contexto no ambiguo con función de censo f_L, entonces el problema consistente en evaluar la función f_L pertenece a* $\mathbf{NC}^2$.

3.4. Funciones de censo de lenguajes libres del contexto ambiguos

El teorema estudiado en la sección anterior afirma que las funciones de censo de los lenguajes libres del contexto no ambiguos pueden ser calculadas eficientemente en paralelo, es natural preguntarse si esto es cierto para todo lenguaje libre de contexto, esto es: dado L un lenguaje libre de contexto, ¿ocurre que $f_L \in \mathbf{FNC}^2$? En esta sección veremos que esto, desafortunadamente, no es cierto: existen lenguajes libres del contexto de ambigüedad finita cuyas funciones de censo son $\#P_1$ completas.

Observación 3.3 *Dada $\mathcal{M}$, una TM no determinística (NDTM), es posible construir una NDTM $\mathcal{M}'$ equivalente (en el sentido de que reconoce el mismo lenguaje) y tal que cualquier computación finita de $\mathcal{M}'$ es aceptante y consiste de un número impar de transiciones.*

Definición 3.15 *Si $\mathcal{M}$ es una NDTM, una* configuración *de $\mathcal{M}$ (también llamada* descripción instantánea*) es una tripla $C = (q, p, w)$ donde $q \in Q$ es el estado en que se encuentra $\mathcal{M}$, $p \in \mathbb{N}$ es la posición de la cabeza lectora y $w \in \Gamma^*$ es el contenido de la cinta de entrada.*

Sin pérdida de generalidad, suponga que $\mathcal{M}$ es una NDTM tal que toda computación finita es aceptante y consiste de un número impar, digamos $2n-1$, de transiciones. Entonces, toda computación aceptante de $\mathcal{M}$ se puede codificar por una sucesión de descripciones instantáneas $w_1, w_2, \ldots, w_{2n}$ donde w_1 codifica una configuración inicial, w_{2n} codifica una configuración final y para todo $i = 1, \ldots, 2n-1$ se tiene que $w_i \vdash_{\mathcal{M}} w_{i+1}$. Representaremos una computación aceptante $w_1, w_2, \ldots, w_{2n}$ de $2n-1$ transiciones de $\mathcal{M}$ por la palabra

$$w_1 \# w_2^R \# w_3 \# w_4^R \# \cdots \# w_{2n-1} \# w_{2n}^R \# \# \tag{3.7}$$

donde # es un símbolo especial que no pertenece ni al conjunto de estados ni al alfabeto de $\mathcal{M}$ y para cada $i = 1, \ldots, n$, la expresión w_{2i}^R denota *el reverso* de la cadena w_{2i}. Llamaremos a una cadena como la de la expresión 3.7, una *descripción alternante* de una computación aceptante de $\mathcal{M}$.

Sean S y H el conjunto de todas las configuraciones iniciales y configuraciones aceptantes de $\mathcal{M}$ (respectivamente). Denotaremos con H^R al conjunto de los reversos de todas las palabras en H. Considere ahora los siguientes lenguajes

$$\begin{aligned} L_0(\mathcal{M}) &= \{w\#u^R\# : w \vdash_{\mathcal{M}} u\}^*.\# \\ L_1(\mathcal{M}) &= S.\#.\{w^R\#u\# : w \vdash_{\mathcal{M}} u\}^*.H^R.\#.\# \end{aligned}$$

Por claridad se ha señalado con un punto bajo (.) la concatenación de lenguajes y por simplicidad se han omitido las llaves para representar el lenguaje $\{\#\}$.

Lema 3.6 *(Lema de Hartmanis [14])*

1. *$L_0(\mathcal{M})$ y $L_1(\mathcal{M})$ son lenguajes libres del contexto determinísticos.*

2. *$L_0(\mathcal{M}) \cap L_1(\mathcal{M})$ es el conjunto de todas las descripciones alternantes que codifican las computaciones aceptantes de $\mathcal{M}$.*

Note que una palabra en $L_0(\mathcal{M})$ tiene la forma $w_1\#u_1^R\#w_2\#u_2^R\#\cdots w_{2t-1}\#u_{2t-1}^R\#w_t\#u_t^R\#\#$ y codifica una secuencia finita de configuraciones $C_1, C_2, C_3, C_4, \ldots, C_{r-1}, C_r$ tales que $C_1 \vdash_{\mathcal{M}} C_2$, $C_3 \vdash_{\mathcal{M}} C_4$, …, $C_{r-1} \vdash_{\mathcal{M}} C_r$; dada esta secuencia, existe una palabra en $L_1(\mathcal{M})$ que codifica una segunda secuencia $C_0, C_1, C_2, C_3, C_4, \ldots, C_r, C_{r+1}$ de configuraciones tales que C_0 es una configuración inicial, $C_0 \vdash_{\mathcal{M}} C_1$, $C_2 \vdash_{\mathcal{M}} C_3$, …, $C_r \vdash_{\mathcal{M}} C_{r+1}$ y C_{r+1} es una configuración aceptante. Esto implica que

1. Si $w \in L_0(\mathcal{M}) \cap L_1(\mathcal{M})$ entonces w es la única descripción alternante de una computación aceptante en $\mathcal{M}$.

2. Para cada computación aceptante descrita por una secuencia de configuraciones C_0, C_1, C_2,…, C_r, C_{r+1} existe una única descripción alternante $w \in L_0(\mathcal{M}) \cap L_1(\mathcal{M})$ que la codifica.

De esto se sigue que: existe una biyección entre el lenguaje $L_0(\mathcal{M}) \cap L_1(\mathcal{M})$ y el conjunto de computaciones aceptantes de $\mathcal{M}$.

Note además que al ser $L_0(\mathcal{M})$ y $L_1(\mathcal{M})$ CFL determinísticos, son por lo tanto, CFL no ambiguos [15]; existe entonces una gramática libre del contexto G, tal que $L(G) = L_0(\mathcal{M}) \cup L_1(\mathcal{M})$, más formalmente, si $G_0 = (\Sigma_0, V_0, P_0, S_0)$ y $G_1 = (\Sigma_1, V_1, P_1, S_1)$ con $(V_0 \cap V_1 = \emptyset)$, son gramáticas libres del contexto no ambiguas tales que $L(G_0) = L_0(\mathcal{M})$ y $L(G_1) = L_1(\mathcal{M})$, entonces formamos la gramática libre del contexto $G = (\Sigma, V, P, S)$, con $S \notin V_0 \cup V_1$, donde

- $\Sigma = \Sigma_0 \cup \Sigma_1$
- $V = V_0 \cup V_1 \cup \{S\}$
- $P = \{S \to S_0 \mid S_1\} \cup P_0 \cup P_1$

Claramente, G es una gramática libre del contexto de ambigüedad a lo más 2 [4], de hecho, como $L_0(\mathcal{M}) \cap L_1(\mathcal{M})$ es no vacío, G es de ambigüedad 2.

Proposición 3.5 *Para toda función $f \in \#P_1$, existe un lenguaje libre del contexto L de ambigüedad 2 tal que f es $\#1$-reducible a f_L (la función de censo de L).*

Prueba. *Sea $\mathcal{M}$ una CMT que calcula f en tiempo cn^c (polinomial) para alguna constante $c > 0$. Sea $\mathcal{M}'$ una CMT que calcula f en tiempo polinomial y es tal que para cada $n \in \mathbb{N}$, todas las descripciones alternantes de $\mathcal{M}'$ sobre el input 1^n tienen la misma longitud, digamos $g(n)$ para alguna función polinomial inyectiva g. Entonces*

$$\begin{aligned} f(1^n) = aac_{\mathcal{M}'}(1^n) &= |L_0(\mathcal{M}') \cap L_1(\mathcal{M}') \cap \Omega^{g(n)}| && (3.8) \\ &= f_{L_0 \cap L_1}(1^{g(n)}) = f_{L_0}(1^{g(n)}) + f_{L_1}(1^{g(n)}) - f_{L_0 \cup L_1}(1^{g(n)}) && (3.9) \end{aligned}$$

donde Ω es un alfabeto adecuado, f_{L_0}, f_{L_1}, $f_{L_0 \cap L_1}$ y $f_{L_0 \cup L_1}$ son las funciones de censo de los lenguajes $L_0(\mathcal{M}')$, $L_1(\mathcal{M}')$, $L_0(\mathcal{M}') \cap L_1(\mathcal{M}')$ y $L_0(\mathcal{M}') \cup L_1(\mathcal{M}')$ respectivamente. La ecuación 3.9 es consecuencia inmediata del principio de inclusión-exclusión.

Puesto que $L_0(\mathcal{M}')$ y $L_1(\mathcal{M}')$ son no ambiguos, por el Corolario 3.4, f_{L_0} y f_{L_1} pueden ser calculadas eficientemente, por lo tanto, la función f es $\#1$-reducible a

$f_{L_0 \cup L_1}$, la función de censo del lenguaje libre del contexto $L_0(\mathcal{M}') \cup L_1(\mathcal{M}')$, que es un lenguaje de ambigüedad 2. ■

Intuitivamente, la Proposición 3.5 muestra que las funciones en $\#P_1$ son al menos tan difíciles como las funciones de censo de lenguajes libres del contexto de ambigüedad 2. Podemos hacer más precisa esta afirmación con el enunciado del corolario a continuación.

Corolario 3.5 *Existe un lenguaje libre de contexto L tal que su función de censo f_L es $\#P_1$-completa.*

Prueba. Sea f un problema $\#P_1$ completo, el Lema 3.5 garantiza la existencia de un lenguaje libre de contexto (¡de ambigüedad 2!), digamos L, tal que f es $\#_1$-reducible a f_L. Tenemos entonces que f_L es $\#P_1$ completa y el corolario queda demostrado. ■

3.5. Aproximabilidad

Los resultados de la sección anterior sugieren que la gran mayoría de los lenguajes libres del contexto poseen funciones de censo que no se pueden calcular eficientemente. Cuando nos enfrentamos a un problema de conteo, tenemos dos alternativas, resolver el problema de manera exacta (en caso de ser posible) o resolverlo de manera aproximada (que podría ser más fácil). En esta sección introducimos la noción de $FPTRAS$ (*full polynomial time randomized approximation scheme*) y discutimos brevemente la teoría relacionada. En la siguiente sección mostraremos que si L es un lenguaje libre del contexto de ambigüedad finita, existe entonces un $FPTRAS$ que permite aproximar la función f_L.

3.5.1. El modelo PrRAM

Estudiaremos a continuación el modelo de computación en el que nos apoyaremos para describir los algoritmos de generación aleatoria uniforme, a saber, las *máquinas de acceso aleatorio probabilísticas,* en adelante PrRAM (*probabilistic random access machine*).

Recordamos que una *máquina de acceso aleatorio (RAM)* consiste de una *cinta de entrada* de solo lectura, una *cinta de salida* de solo escritura, una *memoria* y un *programa* (Figura 3.1).

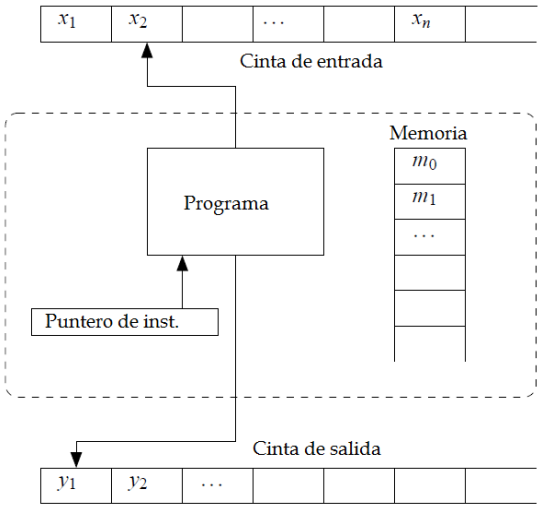

Figura 3.1: Máquina de acceso aleatorio (RAM).

La *cinta de entrada* es una sucesión de celdas, cada una de las cuales puede albergar un entero; cuando un entero es leído desde la cinta de entrada, el cabezal de la cinta se desplaza una celda a la derecha. La *cinta de salida* es una sucesión (potencialmente infinita) de celdas en blanco; cuando una instrucción de escritura es ejecutada, un entero es impreso en la celda donde se encuentre actualmente el cabezal de la cinta de salida y, después de la escritura, el cabezal se desplaza a la siguiente celda a la derecha. La *memoria* consiste en una sucesión de registros $m_0, m_1, \ldots$ cada uno de los cuales puede albergar un entero; no hay cota superior para el número de registros que pueden ser usados.

El *programa* es una secuencia fija de instrucciones (opcionalmente etiquetadas) que no son modificadas durante la ejecución de la RAM. Asumiremos que existen instrucciones de entrada y salida (`read` y `write`) que manipulan las cintas, instrucciones de movimiento de datos desde y hacia la memoria (`load` y `store`), instrucciones aritméticas (`add`, `sub`, `mult` y `div`), instrucciones de manipulación del puntero (`jum`, `jzero` y `jgtz`) y la instrucción `halt` para detener la computación de la RAM.

Para una completa revisión de las instrucciones y de la *semántica* de la computación del modelo RAM, el lector puede consultar la referencia [1].

Cada instrucción opera su argumento según ciertas convenciones (*modo de direccionamiento indirecto*): un argumento de la forma "`=i`" indica el entero $i \in \mathbb{Z}$, "`i`" indica el contenido del registro m_i y "`*i`" denota el contenido del registro m_j donde j es el contenido del registro m_i. En adelante, denotaremos el contenido del registro m_i como $M(i)$.

A manera de ejemplo, la Tabla 3.1 muestra algunas instrucciones y su significado.

Instrucción	**Significado**
`read i`	$M(i)$ toma como valor el entero en la celda actual de la cinta de entrada
`write *i`	Se escribe en la cinta de salida el valor del registro $m_{M(i)}$
`load i`	$M(0)$ toma el valor $M(i)$
`store i`	$M(i)$ toma el valor $M(0)$
`add i`	$M(0)$ toma el valor $M(0) + M(i)$
`div =i`	$M(0)$ toma el valor $\lfloor M(0)/i \rfloor$
`jump =i`	El puntero de instrucción se mueve a la instrucción i-ésima

Tabla 3.1: Instrucciones RAM

Usaremos el *criterio de costo logarítmico* para estimar la complejidad de los algoritmos que pueden ser implementados en este modelo RAM. Sea $\log(i)$ la función logarítmica sobre los enteros definida por

$$\log(i) = \begin{cases} \lfloor \log_2(i) \rfloor + 1 & \text{si} \quad i > 0 \\ 1 & \text{si} \quad i = 0 \end{cases}$$

Para cada instrucción posible del modelo RAM, se define el *costo (logarítmico)* $t(i)$ de acceder el argumento i como $t(= i) = \log(i)$, $t(i) = \log(i) + \log(M(i))$ y $t(*i) = \log(i) + \log(M(i)) + \log(M(M(i)))$.

Observación 3.4 *El lenguaje de alto nivel con el que cuenta el modelo RAM permite el diseño de algoritmos fáciles de entender puesto que son más similares a los algoritmos implementados en los lenguajes de programación de la actualidad. Por otro lado, el criterio de costo logarítmico asumido para el análisis de la complejidad computacional es realista en tanto que es acorde con el modelo de máquinas de*

Turing, es decir, los tiempos de computo de ambos modelos están polinomialmente relacionados.

Recordamos que dos funciones f, g se dice que están polinomialmente relacionadas si existen dos polinomios p, q tales que $f(n) \leq p(g(n))$ y $g(n) \leq q(f(n))$ para todo $n > 0$.

Proposición 3.6 *([26]) Los costos en tiempo (y espacio) del modelo RAM y el modelo de máquinas de Turing están polinomialmente relacionados.*

A fin de obtener un modelo apropiado para algoritmos aleatorios, es necesario introducir un modelo de computación probabilístico. Una *máquina de acceso aleatorio probabilística* (PrRAM) es una RAM dotada de una cinta semi-infinita de solo lectura a la que llamaremos *cinta aleatoria* y una instrucción `rnd` que le permite acceder a ésta (Figura 3.2). Cada celda de la cinta aleatoria contiene un cero o un uno; cuando la instrucción "`rnd a`" es ejecutada, el cabezal de la cinta aleatoria escanea a celdas de la cinta aleatoria, i.e. escanea una secuencia de a bits, esta secuencia de bits es transferida a un registro en la memoria (eventualmente m_0) y es interpretado allí como un entero en notación binaria.

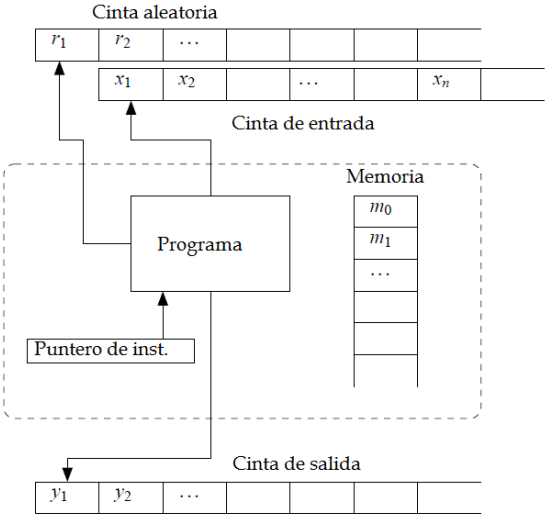

Figura 3.2: Máquina de acceso aleatorio probabilística (PrRAM)

La complejidad computacional de una PrRAM es definida análogamente al modelo RAM, considerando adicionalmente el costo de la instrucción "`rnd a`" que se

define como $t(a)$; aquí el argumento `"a"` toma una de las tres formas del *modo de direccionamiento indirecto* ya mencionado en el modelo RAM. Al número de celdas leídas por la instrucción `rnd` lo llamaremos el *número de bits aleatorios* usados en la computación.

3.6. Generación aleatoria uniforme

En adelante, usaremos el símbolo $\perp$ para denotar un elemento distinguido que no pertenece a algún alfabeto. El símbolo $\perp$ será útil si queremos indicar un valor especial en el `output` de un algoritmo probabilístico, por ejemplo, para indicar una noción natural de *error*.

Notación 3.1 *Si $L \subseteq \Sigma^*$ es un lenguaje formal, usaremos el símbolo L_n para denotar el conjunto*

$$L_n = \Sigma^n \cap L$$

Definición 3.16 *Un algoritmo $\mathcal{A}$ es un* generador aleatorio uniforme *(u.r.g.) para un lenguaje formal $L \subseteq \Sigma^*$ si y solo si, para cada $n > 0$ tal que $f_L(n) > 0$, se tiene lo siguiente*

1. *El output de $\mathcal{A}$ en el input n, denotado $\mathcal{A}(n)$, pertenece al conjunto $L_n \cup \{\perp\}$*
2. *Para toda palabra $w \in L_n$, $\Pr[\mathcal{A}(n) = w \mid \mathcal{A}(n) \neq \perp] = \frac{1}{|L_n|}$*
3. *$\Pr[\mathcal{A}(n) = \perp] < \frac{1}{4}$*

Si $\mathcal{A}$ es un generador aleatorio uniforme que opera en tiempo polinomial en n, diremos que $\mathcal{A}$ es un generador aleatorio uniforme de tiempo polinomial.

La constante $\frac{1}{4}$ en el numeral 3 de la definición anterior puede ser reemplazada por cualquier número positivo estrictamente menor que 1 como lo muestra el lema a continuación.

Lema 3.7 *Sea L un lenguaje formal L y sea $\mathcal{A}$ un algoritmo para el cual los numerales 1 y 3 de la Definición 3.16 se satisfacen, suponga además que* $\Pr[\mathcal{A}(n) = \perp$

$] < \delta$, *para algún* $0 < \delta < 1$. *Entonces, para todo* $0 < \delta' < 1$ *existe un algoritmo* $\mathcal{A}'$ *para el cual los numerales 1 y 3 de la Definición 3.16 se satisfacen y adicionalmente se satisface la desigualdad* $\Pr[\mathcal{A}'(n) = \perp] < \delta'$. *Más aún, si* $\mathcal{A}$ *trabaja en tiempo* $t_{\mathcal{A}}(n)$ *y usa* $R_{\mathcal{A}}(n)$ *bits aleatorios, entonces* $\mathcal{A}'$ *trabaja en tiempo* $O(t_{\mathcal{A}}(n))$ *y usa* $O(R_{\mathcal{A}}(n))$ *bits aleatorios.*

Prueba. *Si* $\delta \leq \delta'$ *la afirmación es trivialmente cierta. En otro caso, sea* AMP *el Algoritmo 3.1.*

Algoritmo 3.1 AMP: amplificador de probabilidades

Input: n

1: $s \leftarrow \perp$, $i \leftarrow 0$
2: **while** $i < \left\lceil \frac{\log \delta'}{\log \delta} \right\rceil$ and $s = \perp$ **do**
3: $\quad i \leftarrow i + 1$
4: $\quad s \leftarrow \mathcal{A}(n)$
5: **return** s

Si n *es un entero tal que* $f_L(n) > 0$, *entonces*

$$\Pr[\mathcal{A}'(n) = \perp] = (\Pr[\mathcal{A}(n) = \perp])^{\left\lceil \frac{\log \delta'}{\log \delta} \right\rceil} < \delta^{\left\lceil \frac{\log \delta'}{\log \delta} \right\rceil} < \delta'$$

Por otro lado, para cada $w \in L_n$ *se tiene que*

$$\Pr[\mathcal{A}'(n) = w \mid \mathcal{A}'(n) \neq \perp] = \Pr[\mathcal{A}(n) = w \mid \mathcal{A}(n) \neq \perp]$$

Las afirmaciones acerca del tiempo de computo y número de bits aleatorios usados por $\mathcal{A}'$ *se siguen inmediatamente de su definición.* ■

Definición 3.17 *Un algoritmo* $\mathcal{A}$ *es un* esquema de aproximación aleatorio *(r.a.s.) para la función de censo* f_L *de un lenguaje formal* $L \subseteq \Sigma^*$ *si y solo si, para cada* $n > 0$ *tal que* $f_L(n) > 0$ *y para cada* $\epsilon \in (0, 1)$, *se tiene lo siguiente*

1. $\mathcal{A}(n, \epsilon) \in \mathbb{Q} \cup \{\perp\}$
2. $\Pr[(1 - \epsilon) f_L(n) \leq A(n, \epsilon) \leq (1 + \epsilon) f_L(n) \mid \mathcal{A}(n, \epsilon) \neq \perp] > \frac{3}{4}$
3. $\Pr[\mathcal{A}(n) = \perp] < \frac{1}{4}$

Más aún, un r.a.s. se dice que es de tiempo completamente polinomial *o (FPTRAS) si este opera en tiempo polinomial en n y $\frac{1}{\epsilon}$.*

La constante $\frac{1}{4}$ del numeral 3 puede ser reemplazada por cualquier número positivo estrictamente menor que uno [17]; también, la constante $\frac{3}{4}$ del numeral 2 puede ser reemplazada por cualquier número en el intervalo $\left(\frac{1}{2}, 1\right)$ [26].

Como ya lo dijimos, en esta sección probaremos que para cada lenguaje libre del contexto de ambigüedad finita existe un $FPTRAS$ para su función de censo. La existencia de algoritmos aleatorios de conteo aproximado (r.a.s.) esta estrechamente ligada a la existencia de generadores aleatorios uniformes (u.r.g.). Esta importante conexión fue descubierta por Jerrum, Valiant y Vazirani en [17]. Tomando nota de esta importante relación, antes de probar que para cada CFL de ambigüedad finita existe un $FPTRAS$ que aproxima su función de censo, mostraremos la existencia de generadores uniformes para tales lenguajes.

Notación 3.2 *Sea $G = (\Sigma, V, P, S)$ una CFG. Supondremos que G está en forma normal de Chomsky y supondremos que V no contiene variables inútiles. Sea A una variable en V. Considere las siguientes definiciones:*

1. $L_A = \{w \in \Sigma^* : A \stackrel{*}{\Rightarrow} w\}$
2. *$\forall w \in \Sigma^*$, el símbolo $d_A(w)$ denota el número de derivaciones $A \stackrel{*}{\Rightarrow} w$ (o equivalentemente, el número de árboles de derivación con raíz en A de la palabra w).*
3. *Dado $l \in \mathbb{N}$, definimos*

$$\begin{aligned} L_A(l) &= \{w \in L_A : |w| = l\} \\ T_A(l) &= \sum_{|w|=l} d_A(w) \end{aligned}$$

4. *P_A el conjunto de reglas de producción de la forma $A \to BC$, con $B, C \in V$*
5. *P_A^1 el conjunto de reglas de producción de la forma $A \to a$, con $a \in \Sigma$.*

Observación 3.5 *Los objetos introducidos en la definición anterior dependen de G, pero para no sobrecargar la notación hemos decidido no usar subíndices o superíndices que indiquen de manera explícita este hecho.*

3.6.1. Generación uniforme en CFL no ambiguos

Los algoritmos de generación uniforme para lenguajes de ambigüedad finita utilizan como una de sus subrutinas más importantes un algoritmo de generación uniforme para lenguajes no ambiguos, este algoritmo, que fue descubierto por Flajolet, Zimmermann y Van Cutsen [11], será denotado con el símbolo GEN (ver Algoritmo 3.2). La variable κ en el algoritmo GEN es un parámetro utilizado para aumentar la probabilidad de éxito.

Algoritmo 3.2 $GEN(A, l)$

Input: $(A, l) \in V \times \mathbb{N}$, donde V es el conjunto de variables de la gramática
Output: $w \in L_A(l) \cup \{\perp\}$

1: $i \leftarrow 0$, $r \leftarrow \perp$, $w \leftarrow \perp$
2: **while** $i < \kappa$ and $r = \perp$ **do**
3: $\quad i \leftarrow i + 1$
4: $\quad$ seleccione aleatoriamente $u \in \{1, \ldots, 2^{\lceil \log T_A(l) \rceil}\}$
5: $\quad$ **if** $u \leq T_A(l)$ **then**
6: $\quad\quad r \leftarrow u$
7: **if** $r \neq \perp$ **then**
8: $\quad$ **if** $l = 1$ **then**
9: $\quad\quad$ Sea $A \to a$ el r-ésimo elemento de P_A^1
10: $\quad\quad w \leftarrow a$
11: $\quad$ **else**
12: $\quad\quad$ calcule el menor elemento $(A \to BC, k) \in P_A \times \{1, \ldots, l-1\}$ tal que

$$\sum_{(A \to DE, h) \leq (A \to BC, k)} T_D(h) T_E(l-h) \geq r \tag{3.10}$$

13: $\quad\quad w_B \leftarrow GEN(B, k)$
14: $\quad\quad w_C \leftarrow GEN(C, l-k)$
15: $\quad\quad$ **if** $w_B \neq \perp$ and $w_C \neq \perp$ **then**
16: $\quad\quad\quad w \leftarrow w_B w_C$
17: **return** w

Observación 3.6 *Es impreciso decir que GEN es un generador uniforme para gramáticas no ambiguas: no se conoce un único GEN para todas las gramáticas no ambiguas; en cambio, existe un generador uniforme por cada gramática no ambigua. Si quisiéramos ser más precisos deberíamos usar el símbolo $GEN\,(G)$ para denotar el algoritmo asociado a la gramática no ambigua G, sin embargo, para evitar recargar la notación y hacer la lectura más fluida, abusaremos de la notación y del lenguaje e insistiremos en seguir usando el símbolo GEN.*

Fijemos una gramática no ambigua $G = (\Sigma, V, P, S)$, el algoritmo GEN requiere que se introduzca un orden lineal sobre el conjunto de reglas de producción de la gramática.

Suponga que G está en forma normal de Chomsky y suponga que cada uno de los conjuntos Σ y V están dotados cada uno de una relación de orden total $\prec_\Sigma$ y $\preceq_V$, respectivamente. Dado $A \in V$ usaremos el símbolo P_A para denotar el conjunto de las reglas en P que tienen a A como la única variable a la izquierda de la regla y dado $i \geq 1$ usaremos el símbolo P_A^i para denotar el conjunto $\{A \to \alpha : |\alpha| = i\}$. Para cada $l > 0$, definiremos un orden $\preceq_l$ sobre el conjunto $P_A^l \times \{1, ..., l-1\}$ (sobre P_A^1 si $l = 1$), el orden $\preceq_l$ se define de la siguiente manera:

- Si $l = 1$, entonces $A \to a \preceq_1 A \to b$ si $a \prec_\Sigma b$.
- Si $l > 1$, entonces $(A \to BC, h) \preceq_l (A \to DE, k)$ siempre que $h < k$, o bien si $h = k$ y $BC \prec_{V^*} DE$, donde $\prec_{V^*}$ es el orden lexicográfico asociado al orden $\preceq_V$.

Nosotros sabemos que la función generatriz

$$G(z) = \sum_{l=0}^{\infty} T_A(l) z^l$$

de T_A es algebraica [7]. Por el Teorema de Comtet, existen polinomios $p_0(X), p_1(X), ..., p_m(X) \in \mathbb{Z}[X]$ tales que para todo l suficientemente grande se satisface la ecuación 3.11

$$p_0(l)T_A(l) + p_1(l)T_A(l-1) + \cdots + p_m(l)T_A(l-m) = 0 \tag{3.11}$$

Es fácil ver que $T_A(l) \in O(r^n)$ para algún $r > 0$, esto implica que los primero n términos $T_A(1), \ldots, T_A(n)$ se pueden calcular en una RAM en tiempo $O(n^2)$ usando el criterio del costo logarítmico. Más aún, para cada $1 \leq l \leq n$ los enteros $\lceil \log T_A(l) \rceil$ pueden ser calculados en tiempo $O(l \log l)$ para un costo total de $O(n^2 \log n)$.

Si suponemos ahora que G es una gramática no ambigua, entonces

- $d_A(w) \leq 1$ para toda $A \in V$ y para toda $w \in \Sigma^*$
- $T_A(l) = |L_A(l)|$ y por lo tanto $T_S(l) = f_L(l)$ para toda $l \in \mathbb{N}$.

En este caso, es posible construir un *u.r.g.* para el lenguaje L_S utilizando el siguiente procedimiento:

Input: $n \in \mathbb{N}$

1. Para todo $1 \leq l \leq n$ y para toda $A \in V$ calcule los $T_A(l)$.
2. Ejecute el algoritmo GEN en el input (S, n) (ver Algoritmo 3.2).

   Nota: Dado (A, l), el input de GEN, *si* $l = 1$, *entonces* GEN *selecciona aleatoriamente un elemento* $A \to a$ *en el conjunto* P_A^1 *y retorna la palabra formada por el único símbolo* a. *Si* $l > 1$, GEN *escoge aleatoriamente un elemento* $(A \to BC, k)$ *en el conjunto* $P_A \times \{1, \ldots, l-1\}$, *y recursivamente hace llamadas en el input* (B, k) *y* $(C, l-k)$ *y retorna la concatenación de las dos palabras asociadas a éstos inputs.*

   El algoritmo GEN *en el input* $(A, l) \in V \times N$ *retorna un elemento* $w \in L_A(l) \cup \{\perp\}$ *tal que (ver [5])*

   a) Para cada $x \in L_A(l)$ *se tiene que*

   $$\Pr[w = x \mid w \neq \perp] = \frac{1}{T_A(l)} = \frac{1}{|L_A(l)|}$$

   b) $\Pr[w = \perp] < \frac{1}{4}$

3. Retorne la palabra w que corresponde al output de GEN, en el input (S, n).

   Nota: *Note que* $w \in L_S(n) \cup \{\perp\}$, *note también que*

a) Para cada $x \in L_S(n) = L_n$ *se tiene que*

$$\Pr[w = x \mid w \neq \perp] = \frac{1}{T_S(n)} = \frac{1}{f_L(n)}$$

es decir, si $w \neq \perp$ *entonces* w *es una palabra de* L_n *generada aleatoriamente de acuerdo con la distribución uniforme sobre* L_n.

b) $\Pr[w = \perp] < \frac{1}{4}$.

No es difícil ver que en nuestro modelo PrRAM de costo logarítmico, el algoritmo descrito toma un tiempo total $O(n^2 \log n)$ y usa $O(n^2 \log n)$ bits aleatorios [5].

Teorema 3.4 *Todo lenguaje libre del contexto no ambiguo admite un generador aleatorio uniforme de tiempo* $O(n^2 \log n)$ *que usa* $O(n^2 \log n)$ *bits aleatorios.*

Observación 3.7 *Note que si* G *es una gramática ambigua (i.e.* $d_A(w) > 1$ *para algún* $A \in V$ *y* $w \in \Sigma^*$*) entonces, en el input* (A, l)*, el algoritmo* GEN *retorna un elemento* $w \in L_A(l) \cup \{\perp\}$ *tal que*

$$\textit{Para cada } x \in L_A(l), \Pr[w = x \mid w \neq \perp] = \frac{d_A(x)}{T_A(l)}$$

este hecho será útil para estudiar la generación aleatoria uniforme para lenguajes libres del contexto de ambigüedad finita, a su vez, este u.r.g. será clave en la construcción de un $FPTRAS$ *para tales lenguajes.*

Observación 3.8 *Note además que es fácil modificar* GEN *(sin cambiar las probabilidades involucradas y sin cambiar sustancialmente el tiempo de computo) si queremos que en lugar de retornar palabras retorne las derivaciones de tales palabras. Esto es así porque la operación clave en los cálculos recursivos de* GEN *es la elección aleatoria de las reglas de producción.*

3.6.2. Generación uniforme para CFL de ambigüedad finita

Dada G una gramática, ambigua o no ambigua, existe un algoritmo $GEN\,(G)$ el cual se comporta como un u.r.g cuando G es no ambigua. Si G es ambigua el

algoritmo $GEN\,(G)$ de Flajolet *et al* no es un generador uniforme, pero podemos modificarlo para convertirlo en un u.r.g cuando G es de ambigüedad finita.

Fijemos una gramática $G = (\Sigma, V, P, S)$, y sea GEN el algoritmo de Flajolet et al asociado a la gramática G. Supondremos que G es una CFG en forma normal de Chomsky y sin variables inútiles, supondremos además que G es de ambigüedad finita, esto es, que existe un $D \in \mathbb{N}$ tal que $d_S(w) \leq D$ para toda $w \in \Sigma^*$. Atendiendo la Observación 3.7, es posible construir un u.r.g. para L_S siguiendo la siguiente idea intuitiva:

1. Calcule la palabra $w = GEN(S, n)$ (Algoritmo 3.2), si $w \neq \perp$ continúe con el paso 2, en otro caso repita el paso 1 un número apropiado (fijo) de veces, si siempre obtiene $w = \perp$ retorne este valor.

2. Calcule el valor $d_S(w)$

3. Construya una moneda parcializada con probabilidad de *cara* igual a $\frac{1}{d_S(w)}$

4. Haga un lanzamiento de la moneda y en caso de obtener *cara* retorne el valor w, en otro caso retorne $\perp$

Note que la construcción y el lanzamiento de una moneda parcializada son fáciles de simular en nuestro modelo PrRAM, además, puesto que cada palabra w en $L_S(n)$ es generada con probabilidad $\frac{d_S(w)}{T_S(n)}$, el uso de la moneda parcializada permite obtener la palabra w con probabilidad $\frac{d_S(w)}{T_S(n)}\frac{1}{d_S(w)} = \frac{1}{T_S(n)} = \frac{1}{f_L(n)}$, es decir, todas las palabras de longitud n en L son generadas con igual probabilidad. Más aún, un valor adecuado del parámetro κ permite amplificar la probabilidad de éxito tanto como sea necesario. Formalmente, el Algoritmo 3.3 describe el procedimiento, donde $mcm(C)$ denota el mínimo común múltiplo del conjunto $C \subset \mathbb{N}$.

Es importante señalar que el entero r puede ser mayor que m, y puesto que $\frac{1}{2} \leq \frac{m}{2^{\lceil \log m \rceil}} \leq 1$, esto implica que toda palabra w generada por *GEN-AmbFinita* es generada con probabilidad $\frac{h}{d_S(w)}$ para algún $\frac{1}{2} \leq h \leq 1$; sin embargo, el valor de h es fijo e independiente de w y por lo tanto, toda palabra w de longitud n es generada con igual probabilidad. Además, señalamos que el valor $d_S(x)$ en la línea 6 del Algoritmo 3.3 es calculado haciendo uso del algoritmo de Earley [10]. El tiempo

Algoritmo 3.3 GEN-AmbFinita(G)

Input: n

Output: $w \in L_n \cup \{\perp\}$

```
1: m ← mcm({1,...,D}), l ← ⌈log m⌉, i ← 0, w ←⊥
2: while i < κD and w =⊥ do
3:     i ← i + 1
4:     x ← GEN(S, n)
5:     if x ≠⊥ then
6:         d ← d_S(x)
7:         seleccione aleatoriamente r ∈ {1,...,2^l}
8:         if r ≤ m/d then
9:             w ← x
10: return w
```

de cómputo empleado en el cálculo de $d_S(x)$ está dado por el Lema 3.8 probado en [5].

Lema 3.8 *Dada una gramática libre del contexto en forma normal de Chomsky y de ambigüedad finita, existe un algoritmo que con input $w \in L$ calcula el número de árboles de derivación de w en tiempo $O(|w|^2 \log |w|)$ y espacio $O(|w|^2)$ usando una PrRAM con criterio de costo logarítmico.*

La prueba es constructiva y los autores exhiben el algoritmo. No es difícil probar que *GEN-AmbFinita* es un u.r.g. para L y que la computación más costosa del algoritmo es, de hecho, el cálculo de $d_S(w)$.

Es fácil probar que existe un algoritmo de tiempo polinomial que permite calcular la función T_S, este hecho será fundamental en el desarrollo de nuestros esquemas de aproximación. Note que si L es un lenguaje no ambiguo y G es una gramática de Chomsky que lo genera y que a su vez es no ambigua (i.e. para todo $w \in L$ se satisface la igualdad $d_S(w) = 1$), entonces las funciones f_L y T_S coinciden, esto es así dado que

$$f_L(n) = \sum_{w \in L_n} 1 = \sum_{w \in L_n} d_S(w) = T_S(n)$$

El algoritmo utilizado en la prueba del Corolario 3.4 es en realidad un algoritmo que permite calcular la función T_S en tiempo $O\left(\log^2 n\right)$ usando un número polinomial de procesadores, (y esto para cualquier gramática libre de contexto). Dado $\mathcal{M}$ el

algoritmo paralelo que calcula la función T_S, siempre es posible obtener de $\mathcal{M}$ un algoritmo secuencial de tiempo polinomial que calcule la misma función.

Teorema 3.5 *Si L es un lenguaje libre del contexto de ambigüedad finita, entonces existe un u.r.g. para L de tiempo $O(n^2 \log n)$ y espacio $O(n^2)$ empleando una PrRAM con criterio de costo logarítmico.*

Observación 3.9 *El Teorema 3.4 puede ser extendido, con un poco de trabajo, a la clase de funciones de censo de los lenguajes libres del contexto de ambigüedad polinomial.*

3.7. FPTRAS para las funciones de censo de los lenguajes libres del contexto de ambigüedad polinomial

En esta sección usaremos la relación existente entre generación aleatoria uniforme y conteo aproximado para exhibir esquemas de aproximación aleatoria ($FPTRAS$) para las funciones de censo de los lenguajes libres del contexto de ambigüedad finita.

Sea G una CFG y sea L el lenguaje generado por G. Recuerde que en la Notación 3.2 (pág. 45) para cada $A \in V$ hemos definido implícitamente las funciones $d_A : L \to \mathbb{N}$ y $T_A : \mathbb{N} \to \mathbb{N}$ como:

$$\begin{aligned} \forall x \in L,\, d_A(x) &= |\{t \in Tree_A : h(t) = x\}| \\ \forall l \in \mathbb{N},\, T_A(l) &= \sum_{|w|=l} d_A(w) \end{aligned}$$

donde $Tree_A$ es el conjunto de todos los árboles de G-derivaciones con raíz en A, y $h : Tree_A \to L$ es la función que a cada árbol $t \in Tree_A$ le asigna la palabra en L derivada mediante t. Note que $h(t)$ es la palabra que se obtiene al leer los símbolos terminales ubicados en las hojas de t (tenga en cuenta que t es un *árbol plantado* lo que implica que las hojas de t están ordenadas).

Antes de presentar el $FPTRAS$ para la función de censo de un CFL de ambigüedad polinomial, presentaremos un Lema que es una forma de la famosa desigualdad de Hoeffding's.

Lema 3.9 *Si $X, X_1, \ldots, X_n$ son variables aleatorias independientes e idénticamente distribuidas que toman valores en el intervalo $[0,1]$. Entonces, para cada $\epsilon > 0$*

$$\Pr\left[(1-\epsilon)E[X] \leq \sum_{i=1}^{n} \frac{X_i}{n} \leq (1+\epsilon)E[X]\right] > 1 - 2e^{-2n(E[X]\epsilon)^2}$$

Teorema 3.6 *Sea L un lenguaje libre del contexto de ambigüedad polinomial explícitamente descrito por una gramática G tal que para cada $w \in L$ se satisface la desigualdad $d(w) \leq O(|w|^D)$, (i.e. G es una gramática de ambigüedad polinomial). Sea Gen un generador aleatorio uniforme para L. Si Gen es computable en tiempo t_{Gen}, T_S es computable en tiempo $t_{T_S}(n)$, h es computable en tiempo t_h y d es computable en tiempo t_d. Existe entonces un esquema de aproximación aleatorio $Aprox(n,\epsilon)$ de tiempo $O(t_{T_S}(n) + \frac{n^{2D}}{\epsilon^2}(t_{Gen}(n) + t_h(n) + t_d(n)))$, donde ϵ es el parámetro que acota el error en la aproximación.*

Prueba. *Por brevedad, definimos $p(n)$, el polinomio que acota la ambigüedad de G, como $p(n) = k_1 n^D + k_2$, donde k_1 y k_2 son constantes enteras tales que para cada $w \in L$ se satisface la desigualdad $d(w) \leq p(|w|)$. Sea $Aprox$ el Algoritmo 3.4 donde $\kappa > 0$ es una constante entera cuyo valor puede ser ajustado previamente.*

Algoritmo 3.4 *$Aprox(n,\epsilon)$)*

Input: n, ϵ

1: $i \leftarrow 0$, $j \leftarrow 0$, $m \leftarrow 0$
2: **while** $i < \kappa \lceil p(n)/\epsilon \rceil^2$ **do**
3: $\quad i \leftarrow i + 1$
4: $\quad t \leftarrow Gen(n)$
5: $\quad$ **if** $t \neq \perp$ **then**
6: $\quad\quad j \leftarrow j + 1$
7: $\quad\quad m \leftarrow m + \frac{1}{d(h(t))}$
8: **if** $j > 0$ **then**
9: $\quad$ **return** $\frac{mT_S(n)}{j}$
10: **else**
11: $\quad$ **return** $\perp$

Por cada entrada al ciclo ***while*** *(línea 2) se ejecutan a lo más una vez los procedimientos Gen, d y h (ver líneas 4 y 7), además, $T_S(n)$ al estar fuera del ámbito de cualquier ciclo es ejecutado a lo más una vez en todo el algoritmo (línea 9). Puesto que el ciclo* ***while*** *es ejecutado $\kappa \left\lceil \frac{p(n)}{\epsilon} \right\rceil^2$ veces y dado que $p(n) \in O(n^D)$, el algoritmo Aprox opera en tiempo $O(t_{T_S} + \frac{n^{2D}}{\epsilon^2}(t_{Gen}(n) + t_h(n) + t_d(n)))$. Solo queda por mostrar que Aprox es un esquema de aproximación aleatorio para la función de censo f_L. Fijemos un n tal que $f_L(n) > 0$ y sea J la variable aleatoria que representa el valor de j al final de la ejecución del Algoritmo Aprox. Entonces, puesto que $\Pr[Gen(n) = \perp] < \frac{1}{4}$, para un κ apropiado se tiene que:*

$$\Pr[Aprox(n) = \perp] = \Pr[J = 0] = (\Pr[Gen(n) = \perp])^{\kappa \left\lceil \frac{p(n)}{\epsilon} \right\rceil^2} < \frac{1}{4}$$

Consideremos ahora el caso $J > 0$. Denotaremos como $Tree_S(n)$ el conjunto de árboles de derivación con raíz en S que producen cadenas de longitud n (i.e. $Tree_S(n) = \{t \in Tree_S : h(t) \in L_n\}$). Definamos ahora

$$\begin{aligned}
\mu &= E\left[\frac{1}{d(h(Gen(n)))} \mid Gen(n) \neq \perp\right] \\
&= \sum_{t \in Tree_S(n)} \frac{1}{d(h(t))} \frac{1}{T_S(n)} \\
&= \sum_{w \in L_n} \left(\frac{1}{T_S(n)} \sum_{t \in h^{-1}(w)} \frac{1}{d(w)} \right) \\
&= \frac{1}{T_S(n)} |L_n| \\
&= \frac{f_L(n)}{T_S(n)}
\end{aligned}$$

Si M es la variable aleatoria que representa los valores asumidos por m al final de la ejecución del algoritmo Aprox y $I(n, \epsilon)$ es el intervalo cerrado $[(1-\epsilon)\mu, (1+\epsilon)\mu]$, el Lema 3.9 nos permite afirmar que

$$\Pr\left[\frac{M}{J} \notin I(n, \epsilon) \mid J = k\right] \leq 1 - \Pr\left[(1-\epsilon)\mu \leq \frac{M}{J} \leq (1+\epsilon)\mu \mid J = k\right] \leq 2e^{-2k(\mu\epsilon)^2}$$

para todo $k > 0$. De aquí, si $N = \kappa \left\lceil \frac{p(n)}{\epsilon} \right\rceil^2$ y $q = \Pr\left[Gen(n) \neq \perp\right] > \frac{3}{4}$, entonces el teorema de Bayes [3] implica que

$$\begin{aligned}
\Pr\left[\frac{M}{J} \notin I(n,\epsilon) \mid J > 0\right] &= \frac{\Pr\left[\frac{M}{J} \notin I(n,\epsilon) \cap J > 0\right]}{\Pr\left[J > 0\right]} \\
&= \frac{\left(\sum_{k=1}^{N} \Pr\left[\frac{M}{J} \notin I(n,\epsilon) \mid J = k\right]\right) \Pr[J = k]}{\Pr\left[J > 0\right]} \\
&\leq \frac{8}{3} \sum_{k=1}^{N} e^{-2k(\mu\epsilon)^2} \binom{N}{k} q^k (1-q)^{N-k} \\
&\leq \frac{8}{3} \sum_{k=1}^{N} \binom{N}{k} \left(\frac{q}{e^{2(\mu\epsilon)^2}}\right)^k (1-q)^{N-k} \\
&< \frac{8}{3} \left(1 - q\left(1 - \frac{1}{e^{2(\mu\epsilon)^2}}\right)\right)^N \\
&\leq \frac{8}{3} \left(1 - \frac{3}{4}\left(\frac{\epsilon}{p(n)}\right)^2\right)^{\kappa \left\lceil \frac{p(n)}{\epsilon} \right\rceil^2}
\end{aligned}$$

la última desigualdad se sigue del hecho de que $1 - \frac{1}{e^x} \geq \frac{x}{2}$ para todo $x \in [0,1]$ y de que $T_S(n) \leq f_L(n)p(n)$.

Finalmente no es difícil ver que para un $\kappa > 0$ apropiado se tiene que

$$\frac{8}{3} \left(1 - \frac{3}{4}\left(\frac{\epsilon}{p(n)}\right)^2\right)^{\kappa \left\lceil \frac{p(n)}{\epsilon} \right\rceil^2} < \frac{1}{4}$$

Ahora bien, puesto que $Aprox(n,\epsilon) = \frac{MT_S(n)}{J}$ (ver línea 9 de Aprox) se verifica que

$$\Pr\left[\frac{M}{J} \in I(n,\epsilon) \mid J > 0\right] = \Pr\left[(1-\epsilon)f_L(n) \leq Aprox(n,\epsilon) \leq (1+\epsilon)f_L(n) \mid Aprox(n,\epsilon) \neq \perp\right]$$

y de aquí que

$$\Pr\left[(1-\epsilon)f_L(n) \leq Aprox(n,\epsilon) \leq (1+\epsilon)f_L(n) \mid Aprox(n,\epsilon) \neq \perp\right] \geq \frac{3}{4}$$

con lo que queda probado que Approx es un esquema de aproximación aleatorio para f_L. ■

Observación 3.10 *Jerrum, Valiant y Vazirani [17] muestran que si L es un lenguaje libre de contexto de ambigüedad estrictamente exponencial existe un esquema de aproximación aleatorio que permite calcular f_L en tiempo cuasipolinomial.*

Capítulo 4

Aplicaciones

En este capítulo, el último de este trabajo, usaremos algunos de los teoremas discutidos previamente para estudiar desde el punto de vista algorítmico dos problemas provenientes de las ciencias aplicadas, en su orden: tomografía discreta y mecánica estadística.

4.1. Aplicaciones en tomografía discreta: conteo de polígonos con área prescrita.

En esta sección consideraremos un problema de conteo delgado relacionado con polígonos y exhibiremos un algoritmo en paralelo que resuelve este problema. Nuestro algoritmo está basado en el método de Schützenberger-Bertoni.

Observación 4.1 *El material estudiado en esta sección está basado en la referencia [20].*

El conteo de polígonos de área prescrita es uno de los problemas de conteo delgado más estudiados puesto que tiene muchas aplicaciones en termodinámica, fisicoquímica y tomografía discreta (ver referencia [2] y las referencias en éste). En tomografía discreta el conteo de polígonos es importante debido a que, en la mayoría de los casos, las instancias de un problema tomográfico dado son ambiguas, esto es, las instancias admiten varias soluciones. Para los usuarios de las imágenes tomográficas

(e.g. médicos), es de interés calcular una solución única (la solución). El conteo de polígonos puede ser usado para reducir el grado de ambigüedad de la instancia dada.

Dado $r \geq 1$ usaremos el símbolo $\mathcal{L}_2^r$ para denotar la grilla rectangular $\mathbb{N} \times [r]$.

Definición 4.1 *Un polígono es un ciclo simple incluido en $\mathcal{L}_2^r$ que no contiene subciclos propios.*

Definición 4.2 *Dado un polígono P, el área de P, denotada $A(P)$, es igual al número de celdas encerradas por éste.*

Sea I_0 el conjunto $\{(1,k) : k \leq r\}$. El problema $\#Poly[r]$ es el problema de conteo delgado definido por:

Problema 4.1 *($\#Poly[r]$, Conteo de polígonos en grillas rectangulares de altura r)*

- *Input:* 1^n, *donde* $n \in \mathbb{N}$.
- *Problema: calcular el número de polígonos de área n que intersecan el conjunto I_0.*

Observación 4.2 *Dado $n, r \geq 2$, existen infinitos polígonos de área n contenidos en $\mathcal{L}_2^r$, y todo polígono es la traslación de un polígono que interseca I_0.*

Nosotros probamos el siguiente teorema.

Teorema 4.1 *Dado un entero positivo r, el problema de conteo delgado $\#Poly[r]$ puede ser resuelto en tiempo $O\left(\log^2(n)\right)$ usando un número polinomial de procesadores.*

Note que para obtener una prueba del teorema 4.1 es suficiente mostrar que, si fijamos $r \geq 1$, podemos calcular en tiempo $2^{O(r)}$ un autómata de pila determinístico $\mathcal{M}_r$ que satisfaga:

$$\forall n \geq 1, \#Poly[r](n) = f_{\mathcal{M}_r}(2n+1)$$

donde $\#Poly[r](n)$ denota el número de polígonos de área n y $f_{\mathcal{M}_r}$ denota la función de censo del lenguaje $L(\mathcal{M}_r)$, el lenguaje reconocido por el autómata $\mathcal{M}_r$.

La construcción de $\mathcal{M}_r$ está basada en la codificación de polígonos como palabras sobre un alfabeto adecuado.

Desde ahora, fijaremos $r \geq 1$. Una *porción bilateral* es un grafo en la grilla isomorfo a $\{0,1\} \times [r]$. Las *porciones izquierdas* son los subgrafos de porciones bilaterales obtenidas al eliminar todas las aristas verticales de la forma $\{(1,x),(1,x+1)\}$.

Los *patrones izquierdos* (*patrones*, para abreviar) son los tipos de isomorfismos de los subgrafos de un patrón izquierdo, note que cualquier par de porciones izquierdas de altura r son isomorfas.

Sea $\langle n \rangle$ el conjunto $\{0,...,n\}$, note que dado un polígono γ contenido en la grilla $\langle n \rangle \times [r]$, el polígono γ es la concatenación de una secuencia $p_1...p_n$ de n patrones tales que p_1 es no vacío (p_1 contiene al menos una arista horizontal) y todas las aristas que ocurren en p_n son aristas verticales localizadas en la columna izquierda de p_n. Note también que todo polígono de area n y altura r puede considerarse como contenido en $\langle n \rangle \times [r]$. Lo anterior sugiere que podemos codificar los polígonos de area n como palabras de longitud $n+1$ cuyos caracteres son patrones.

Lema 4.1 *Dado $r \geq 1$, existen a lo más 2^{2r} patrones distintos.*

El lema anterior es consecuencia de que todo patron es un subconjunto del conjunto de aristas de una porción izquierda. Esto a su vez implica que podemos enumerar la colección de todos los patrones en tiempo $2^{O(r)}$ (en tiempo constante dado que r es fijo), usaremos el símbolo $\mathcal{P}_L$ para denotar el conjunto de los patrones izquierdos.

Dados p y q dos patrones, podemos pensar de la concatenación pq de estos dos patrones como en un subgrafo de la grilla $\{0,1,2\} \times [r]$. El conjunto de los nodos de pq que están localizados sobre el conjunto $\{1\} \times [r]$ será llamado el *núcleo* de pq (o la 1-fibra de pq), y será denotado con el símbolo $\varsigma(pq)$. Dado $i \in \{0,1,2,3,4\}$ definimos una función $\alpha_i : \mathcal{P}_L \times \mathcal{P}_L \to \mathbb{N}$ de la siguiente manera:

$$\alpha_i(p,q) = \left|\left\{v \in \varsigma(pq) : \deg_{pq}(v) = i\right\}\right|$$

donde $\deg_{pq}(v)$ denota el grado de v como nodo de pq. Adicionalmente definimos funciones $\beta_4, \beta_3, ..., \beta_0 : \mathcal{P}_L \to \mathbb{N}$, donde dado $i \in \{0,1,2,3,4\}$ la función β_i cuenta el número de nodos de grado i ubicados sobre la 0-fibra del patron siendo evaluado.

Note que nosotros podemos calcular en tiempo $2^{O(r)}$ las tablas de cada una de las funciones $\alpha_0, ..., \alpha_4, \beta_0, ..., \beta_4$. Dados p, q dos patrones, diremos que el par (p, q) es *admisible* si y solo si se satisfacen las ecuaciones

$$\alpha_1(p,q) = \alpha_3(p,q) = \alpha_4(p,q) = 0$$

Usaremos el símbolo $\mathcal{I}$ para denotar el conjunto de los pares admisibles, note que $\mathcal{I}$ puede ser calculado en tiempo $2^{O(r)}$.

Observe que dado γ un subgrafo de $\langle n \rangle \times [r]$, el subgrafo γ es un polígono si y solo si todos los nodos de γ tienen grado dos (como nodos de γ) y γ no contiene ciclos propios. La prueba del lema a continuación es muy fácil y por ello se omite.

Lema 4.2 *Una secuencia de patrones $p_1 p_2 ... p_{n+1}$ codifica un polígono que interseca el conjunto I_0 si y solo si:*

1. *p_1 es no vacío, todas las aristas que ocurren en p_{n+1} son aristas verticales localizadas sobre la columna izquierda de p_{n+1}, $\beta_3(p_1) = \beta_1(p_1) = 0$ y para todo $i \leq n$ se tiene que el par (p_i, p_{i+1}) es admisible.*

2. *Todos los nodos que ocurren en $p_1 p_2 ... p_{n+1}$ tienen grado 2.*

3. *El grafo codificado por $p_1 p_2 ... p_{n+1}$ no contiene ciclos propios.*

Una *n-cadena de patrones* es una secuencia $p_1 p_2 ... p_{n+1}$ que satisface las condiciones impuestas en el enunciado del lema anterior. La prueba del lema a continuación es muy fácil y por ello se omite.

Lema 4.3 *Existe una biyección entre el conjunto de las n-cadenas de patrones y el conjunto de los polígonos incluidos en $\langle n \rangle \times [r]$ que intersecan el conjunto I_0.*

Dado P un polígono, si P es el polígono codificado por $p_1 p_2 ... p_{n+1}$, entonces el area de P, que denotamos con el símbolo $A(P)$, es igual a $\sum_{i \leq n+1} A^*(p_i)$, donde dado p un patron definimos $A^*(p)$, el area de p, como sigue:

Sea $h_1, ..., h_t$ la lista ordenada de las y-coordenadas (alturas) de los lados horizontales que ocurren en p. Podemos suponer que t es un número par igual a $2m$.

Definimos $A^*(p)$ como $\sum_{i=0}^{m-1}(h_{t-2i}-h_{t-2i-1})$. Si p es un patron vacío (que no contiene aristas horizontales) definimos $A^*(p)=0$. Sea $A^*{:}\mathcal{P}_L \to \mathbb{N}$ la función $p \mapsto A^*(p)$, note que una tabla para A^* puede ser calculada en tiempo constante.

Estamos listos para definir nuestra codificación de polígonos de un area dada, para tal fin usaremos el alfabeto $\Sigma = \{p : p \in \mathcal{P}_L\} \cup \{1\}$. Sea $L_{poly(r)}$ el lenguaje

$$L_{poly(r)} = \{1^n p_1 ... p_{n+1} \in \Sigma^* : n \geq 1 \ \& \ \Psi\}$$

donde Ψ es la sentencia definida como la conjunción de las siguientes condiciones:

1. p_1 es no vacío; $\beta_3(p_1) = \beta_1(p_1) = 0$; todos los lados que ocurren en p_{n+1} son lados verticales contenidos en la columna izquierda de p_{n+1}; para todo $i \leq n$ se tiene que (p_i, p_{i+1}) es un par admisible y todos los nodos que ocurren en $p_1 p_2 ... p_{n+1}$ tienen grado dos.

2. La n-cadena de patrones $p_1 p_2 ... p_{n+1}$ no permite la creación de ciclos propios.

3. $A(p_1 p_2 ... p_{n+1}) = n$.

Sea $r \geq 1$ y sea $\mathcal{A}_r$ el conjunto de todos los *subgrafos finitos y acíclicos de* $\mathbb{N} \times [r]$ cuyo grado total es acotado por 2 (i.e. tales que si v es un nodo, el grado de v es menor o igual que 2). El lema a continuación es un lema técnico que será clave en la prueba de los dos teoremas principales de este capítulo.

Lema 4.4 *Dado $r \geq 1$, es posible calcular en tiempo $2^{O(r)}$ un autómata finito $\mathcal{M}_r$ que reconoce el conjunto $\mathcal{A}_r$.*

Prueba. Lo que nosotros probaremos es que el lenguaje

$$\Omega_r = \{p_1 ... p_n \in \Sigma^* : \Psi\}$$

es regular, donde Ψ es igual a la conjunción de las siguientes condiciones:

1. Todos los lados que ocurren en p_n son lados verticales localizados sobre la columna izquierda de p_n, y dado $i \leq n-1$ se tiene que el par (p_i, p_{i+1}) es un par admisible.

2. $\beta_1(p_1) + \sum_{i \leq n-1} \alpha_1(p_i, p_{i+1}) = 2$ y se satisfacen las ecuaciones $\beta_0(p_1) = \beta_3(p_1) = 0$.

3. La cadena $p_1 p_2 ... p_n$ prohíbe la creación de ciclos y no existen nodos en $p_1 p_2 ... p_n$ de grados tres y cuatro.

Podemos expresar Ψ como $\Psi_1 \wedge \Psi_2$, donde Ψ_1 es la conjunción de las dos primeras condiciones y Ψ_2 es la sentencia:

La cadena $p_1 p_2 ... p_n$ prohíbe la creación de ciclos y no existen nodos de grado tres o cuatro contenidos en $p_1 p_2 ... p_n$

Definimos dos lenguajes

$$\Omega_1 = \{p_1 ... p_n \in \Sigma^* : \Psi_1\}$$

y

$$\Omega_2 = \{p_1 ... p_n \in \Sigma^* : \Psi_2\}$$

Y observamos que $\Omega_r = \Omega_1 \cap \Omega_2$. Recuerde que la intersección de lenguajes regulares es regular. Lo anterior implica que es suficiente, para nuestros propósitos, probar que Ω_1 y Ω_2 son lenguajes regulares. Primero mostraremos que Ω_1 es regular, mostraremos que es posible construir en tiempo $2^{O(r)}$ un autómata de estado finito $\mathcal{M}_1$ que reconoce el lenguaje Ω_1.

Recuerde que nosotros podemos calcular los conjuntos $\mathcal{P}_L$ e $\mathcal{I}$; y las tablas de las funciones α_0, α_1, α_3, α_4, β_0, β_1 y β_3 en tiempo $2^{O(r)}$. Entonces, antes de construir el autómata podemos calcular estas tablas e incorporarlas en la función de transición del autómata que las usará como *lookup tables*. Sea $\mathcal{M}_1$ el autómata de estado finito (Q, q_0, F, δ) definido por:

1. $Q = \{(q, i) : q \in \Sigma \ \& \ 0 \leq i \leq 2 \ \} \cup \{q_0, q_r, q_a\}$.

2. $F = Q - \{q_r\}$.

3. Sea $p_1 ... p_n$ un input de $\mathcal{M}_1$. La función de transición δ se define de la siguiente manera:

a) $\delta(q_0, p_1) = (p_1, \beta_1(p_1))$ si $\beta_0(p_1) = \beta_3(p_1) = 0$ y $\beta_1(p_1) = 0$, en otro caso $\delta(q_0, p_1) = q_r$

b) Dado $i \leq n-1$, se tiene que $\delta((p_i, k), p_{i+1}) = q_r$, siempre que alguna de las siguientes condiciones sea satisfecha:

- $(p_i, p_{i+1}) \notin \mathcal{I}$.
- $k + \alpha_1(p_i, p_{i+1}) \geq 1$.

c) Sea $i \leq n-1$. Si $(p_i, p_{i+1}) \in \mathcal{I}$ y $k + \alpha_1(p_i, p_{i+1}) = 0$, entonces

$$\delta((p_i, k), p_{i+1}) = (p_{i+1}, k + \alpha_1(p_i, p_{i+1}))$$

d) Si $\alpha_1(p_{n-1}, p_n) + k = 0$ y p_n no contiene aristas horizontales, entonces

$$\delta((p_n, k), \square) = q_a$$

e) Si $\alpha_1(p_{n-1}, p_{n+1}) + k \neq 0$ o p_n contiene aristas horizontales, entonces

$$\delta((p_n, k), \square) = q_r$$

Note que el autómata $\mathcal{M}_1$ simplemente verifica que la cadena $p_1...p_n$ es una cadena de patrones compatibles tales que el número de nodos de grado uno, tres y cuatro es igual a 0 y tal que todas las aristas en p_n son aristas verticales contenidas en la columna izquierda de p_n. Es fácil (pero tedioso) verificar que el automata $\mathcal{M}_1$ reconoce el lenguaje Ω_1. Tenemos que estimar el tiempo de computo empleado en la construcción de $\mathcal{M}_1$. Note que el tiempo de computo esta acotado superiormente por $2^{O(r)}$, dado que las tablas que este emplea pueden ser calculadas en tiempo $2^{O(r)}$ y además $|Q| \in 2^{O(r)}$.

Para terminar con la prueba debemos mostrar que Ω_2 también es un lenguaje regular. Presentaremos una descripción esquemática de un autómata de estado finito, digamos $\mathcal{M}_2$, que reconoce el lenguaje Ω_2. Mostraremos que si fijamos $n \geq 1$, podemos usar un autómata de estado finito $\mathcal{M}_2$ para reconocer los grafos acíclicos y de *grado total* acotado por dos contenidos en $[n] \times [r]$ (el grado total de un grafo G es igual a $\max_{v \in V(G)} \{\deg(v)\}$)

Una *c-estructura* es un subgrafo de $[n] \times [r]$ constituido por una secuencia vertical de aristas ubicadas a la izquierda, y aristas horizontales ubicadas en los extremos inferior y superior de la secuencia y que se extienden a la derecha de la mencionada secuencia vertical (i.e. son subgrafos con forma de c). Note que:

1. Todo ciclo empieza con una c-estructura, esto es: El patron ubicado al extremo izquierdo del ciclo contiene una c-estructura.

2. Las c-estructuras pueden ser fácilmente identificadas usando un autómata de estado finito.

Podemos pensar en una c-estructura como en una alerta que nos advierte de la posible emergencia de un ciclo. Esto implica que debemos guardar información referente a las c-estructuras que han sido observadas con anterioridad. Para lograr tal objetivo, debemos hacer seguimiento de las trayectorias que surgen de las c-estructuras previamente observadas, esto será posible puesto que:

1. Dado w, un input de $\mathcal{M}_2$, y dado $t \leq |w|$, existen a lo más $\frac{r}{2}$ parejas de trayectorias en el instante t que han sido originadas en c-estructuras previamente observadas que podrían dar lugar a ciclos.

2. La única información que tenemos que guardar, con el fin de controlar la evolución de un par de trayectorias *peligrosas*, son las y-coordenadas de los nodos donde las trayectorias encuentran la columna derecha del patron izquierdo que se está escaneando.

Dado i un entero positivo menor que $\frac{r}{2}$, usaremos el símbolo P_i para denotar el conjunto

$$\{((a_1, b_1), ..., (a_i, b_i)) : \ \Phi\}$$

donde Φ es la condición:

$a_1, ..., a_i, b_1, ..., b_i \in \{1, ..., r\}$ son diferentes dos a dos y $a_1 \lneqq b_1; ...; a_i \lneqq b_i$.

Sea P el conjunto de estados del autómata $\mathcal{M}_2$, el conjunto P es esencialmente igual a $\bigcup_{0 \leq i \leq \frac{r}{2}} P_i$. Sea $p = ((a_1, b_1), ..., (a_i, b_i))$ un elemento de P, y supongamos que

p es el estado de $\mathcal{M}_2$ en el instante t. El estado p está informándonos que las i trayectorias peligrosas que están saliendo del patron izquierdo que se está explorando en el instante t, lo están haciendo a través de los pares de nodos $(a_1, b_1), ..., (a_i, b_i)$. La función de transición de $\mathcal{M}_2$, denotada por ρ, es definida de modo tal que nos permita seguir la pista de la evolución de aquellas trayectorias. Considere como ejemplo la siguiente situación:

1. El estado del autómata $\mathcal{M}_2$, en el instante t, es igual a $((a_1, b_1), (a_2, b_2), (a_3, b_3))$.

2. $a_1 \leqq b_1 \leqq a_2 \leqq b_2 \leqq a_3 \leqq b_3$.

3. La trayectoria cuya y-coordenada es igual a $a_1 \geq 2$, es extendida con una arista vertical hacia abajo y luego con una arista horizontal.

4. La trayectoria cuya y-coordenada es igual a b_1 es extendida con una arista vertical hacia arriba y luego con una arista horizontal.

5. La trayectoria cuya y-coordenada es igual a a_2 es extendida con una arista vertical hacia abajo y luego con una arista horizontal. Además, suponemos que $a_2 - b_1 = 3$.

6. La trayectoria cuya y-coordenada es igual a b_2 es extendida con una arista vertical hacia arriba y luego con una arista horizontal.

7. Las trayectorias restantes son extendidas con aristas horizontales. Además, suponemos que $a_3 - b_2 \geq 2$.

8. Una nueva c-estructura es observada, ésta está conformada por dos aristas horizontales cuyas y-coordenadas son iguales a $k + 3$ y $k \geqq b_3$, y por tres aristas verticales sobre la columna izquierda que unen los nodos localizados a altura k y $k + 3$.

Dada esta información, tenemos que el estado del autómata $\mathcal{M}_2$, en el instante $t + 1$, es igual a

$$((a_1 - 1, b_1 + 1), (a_2 - 1, b_2 + 1)(a_3, b_3), (k, k + 3))$$

Identificamos la evolución de una c-estructura con sus correspondientes parejas de seguimiento (*tracking pairs*). Supongamos que $\mathcal{M}_2$ está leyendo la palabra w, y supongamos que la pareja (a, b) pertenece al estado de $\mathcal{M}_2$ en el instante t. Existen tres posibilidades para la pareja (a, b), en el instante $t + 1$: (i) la pareja (a, b) se fusiona con otra pareja del instante t; (ii) la pareja (a, b) sobrevive o (iii) la pareja (a, b) se *desvanece*. Decimos que la pareja (a, b) se desvanece en el instante $t + 1$ si y solo si w_{t+1}, el $(t + 1)$-ésimo patron de la palabra que se está leyendo, contiene una cadena de aristas verticales que unen los nodos localizados a altura a y b. Si una pareja se desvanece, un ciclo ha sido creado (detectado). Por otro lado, si dos parejas se fusionan, esta fusión da lugar a un ciclo si y solo si las parejas están *anidadas*, esto es: si la pareja (a, b) y (c, d) se funden en el instante $t + 1$, y la desigualdad $a \leqq c \leqq d \leqq b$ se verifica; cuando esto ocurre, un ciclo ha sido creado (detectado). Usaremos el término *vectores-pareja* para denotar la tupla de parejas contenida en el estado actual del autómata $\mathcal{M}_2$, note que la evolución de parejas define un álgebra de *vectores-pareja* la cual es finita y puede ser pre-calculada y codificada dentro de la función de transición de $\mathcal{M}_2$.

Llegados a este punto podemos afirmar que hemos discutido las características claves de $\mathcal{M}_2$, el lector debe estar completamente convencido de que Ω_2 es un lenguaje regular, y que un autómata de estado finito que reconozca Ω_2 puede ser calculado en tiempo $2^{O(r)}$.

Podemos ahora afirmar que un autómata de estado finito que reconozca Ω_r puede ser calculado en tiempo $2^{O(r)}$. ■

Teorema 4.2 *$L_{poly(r)}$ es un lenguaje libre del contexto no ambiguo y podemos calcular en tiempo $2^{O(r)}$ un autómata de pila determinístico $\mathcal{M}_{poly(r)}$ que reconoce $L_{poly(r)}$.*

Prueba. (*Esbozo de la prueba*) Es momento de colocar las piezas juntas. Queremos probar que el lenguaje

$$L_{poly(r)} = \{1^n p_1 ... p_{n+1} \in \Sigma^* : n \geq 1 \ \& \ \Psi\}$$

es un lenguaje libre del contexto no ambiguo, donde Ψ es la conjunción de las tres condiciones listadas anteriormente. Podemos escribir Ψ como $\Psi_1 \wedge \Psi_2 \wedge \Psi_3$, donde

Ψ_1 es la primera condición, Ψ_2 es la segunda condición y Ψ_3 es la última. Definimos los lenguajes L_1, L_2 y L_3, donde dado $i \leq 3$

$$L_i = \{1^n p_1 ... p_{n+1} \in \Sigma^* : n \geq 1 \ \& \ \Psi_i\}$$

Observe ahora que $L_{poly(r)} = L_1 \cap L_2 \cap L_3$. Recordamos que la intersección de un número finito de lenguajes regulares con un lenguaje libre del contexto determinístico da lugar a un lenguaje libre del contexto determinístico, recordamos además que los lenguajes libres del contexto determinísticos son no ambiguos. Por lo tanto, es suficiente mostrar que L_1 y L_2 son regulares y que L_3 es un lenguaje libre del contexto determinístico. Podemos calcular en tiempo $2^{O(r)}$ un autómata de estado finito $\mathcal{M}_1$ que reconozca L_1. El Lema 4.4 nos permite calcular, en tiempo $2^{O(r)}$, un autómata de estado finito $\mathcal{M}_2$ que reconoce L_2. Mostraremos ahora que L_3 es un lenguaje libre del contexto determinístico. Daremos una breve descripción del autómata de pila determinístico $\mathcal{M}_3$ que reconoce L_3. La tabla A^* puede ser calculada en tiempo constante, este hecho nos permite incorporar la función A^* dentro de la función de transición de $\mathcal{M}_3$. Sea w un input de $\mathcal{M}_3$. El autómata $\mathcal{M}_3$, en el input w, opera como sigue:

1. Si el primer símbolo es diferente de 1 el automata rechaza su input.

2. Cada vez que $\mathcal{M}_3$ lee un 1, escribe este 1 en la pila. Si $\mathcal{M}_3$ encuentra un factor $x1$ tal que $x \neq 1$, entonces $\mathcal{M}_3$ rechaza w.

3. Cada vez que $\mathcal{M}_3$ lee un símbolo (p, s), suprime $A^*(p)$ símbolos de su pila.

$\mathcal{M}_3$ acepta w si y solo si la computación de $\mathcal{M}_3$, en el input w, finaliza con la pila vacía y no ocurre que en algún instante de la computación, la pila no contenía suficientes símbolos para ser suprimidos.

Es claro que $\mathcal{M}_3$ puede ser calculado en tiempo $2^{O(r)}$. Por lo tanto, tenemos que existe un autómata de pila determinístico que reconoce $L_{poly(r)}$ y este autómata puede ser calculado en tiempo $2^{O(r)}$. ■

4.2. Aplicaciones en Mecánica Estadística

Observación 4.3 *El material estudiado en esta sección se basa en la referencia [19].*

Las propiedades cualitativas de algunos modelos de la mecánica estadística están completamente codificados dentro de una función discreta llamada *la función de partición del modelo [28].* Muchas cantidades que describen la dinámica y estructura de aquellos modelos, como por ejemplo la *energía libre* del sistema, pueden ser calculadas a partir de su función de partición, así, resolver un modelo discreto de la mecánica estadística significa calcular su función de partición. La mayoría de la veces, la funciones de partición están definidas como problemas de conteo [32]. En esta sección consideraremos el *modelo de caminos que se auto-evitan* (*self-avoiding walk model).* Mencionaremos algunos hechos relacionados con la complejidad computacional de calcular la función de partición de este modelo y probaremos que existe un algoritmo de tiempo $O\left(\log^2(n)\right)$ que opera en paralelo y que calcula el número de caminos que se auto-evitan que están contenidos en grillas de altura fija.

La relación entre los caminos que se auto-evitan y las cadenas de polímeros está basada en la *conformational statistics of polymer chains* desarrollada por S. Edwards y sus colaboradores, quienes muestran que el modelo de caminos que se auto-evitan puede ser usado como un *lattice model* que describe y realiza las relaciones y las afirmaciones básicas de la *teoría del volumen excluido para cadenas de polímeros* (más información puede ser encontrada en [9]), la cual es la teoría termodinámica de uso en el estudio de la organización espacial de los polímeros (ver [12]).

Así, tenemos que un buen entendimiento del comportamiento de la función de partición para el modelo bidimensional de caminos que se auto-evitan aporta información profunda relativa a la evolución de las cadenas de polímeros.

Consideraremos un modelo restringido de cadenas de polímeros: el modelo de cadenas de polímeros contenidos en grillas de altura fija, el cual es importante en el estudio de proteínas globulares [16]. El *lattice-model* que corresponde a este modelo físico es el *modelo bidimensional de caminos que se auto-evitan restringidos a tiras de altura fija (model of two dimensional self-avoiding walks constrained to lattice strips of fixed height).* Probaremos en esta sección que la función de partición de este último

modelo puede ser calculada en tiempo $O\left(\log^2(n)\right)$ empleando un número polinomial de procesadores.

Recordamos que el problema $\#SAW$ es el *problema de conteo delgado* definido por

Problema 4.2 *($\#SAW$, el problema de caminos que se auto-evitan)*

Input: 1^n, *donde* $n \in \mathbb{N}$.

Problema: *Calcule el número de caminos simples (de cualquier longitud) contenidos en grillas cuadradas de orden* n.

A pesar de la cantidad de trabajo realizado relacionado con este problema, pocos resultados son conocidos. No sabemos de la existencia de fórmulas cerradas para la función de conteo que codifica el problema, no sabemos de la existencia de algoritmos eficientes que resuelvan el problema y no sabemos de la existencia de pruebas de dureza que podrían explicar, hasta cierto punto, la intratabilidad del mismo. Randall y Sinclair han encontrado un $FPTRAS$ para el problema $\#SAW$ [23]. El algoritmo Randall-Sinclair puede ser considerado como el único resultado importante obtenido hasta ahora relacionado con la complejidad computacional de $\#SAW$.

Si modificamos algunos parámetros en la definición del problema, podemos obtener una versión factible (*feasible*) de este. Algunas versiones factibles de $\#SAW$ han sido identificadas y estudiadas. Es conocido que, por ejemplo, el conteo de *up-side* y *spiral self-avoiding walks* puede ser realizado en tiempo lineal [21]. Klein [18] y Wall-Klein [31] estudiaron un modelo relacionado, *The Model of Self-avoiding Walks constrained to Lattice Strips of Fixed Height*, este modelo es importante en el estudio de proteínas globulares y largas cadenas de polímeros [16]. Estudiaremos la función de partición de este problema, denotada por $\#SAW_r$, y exhibiremos un algoritmo que calcula $\#SAW_r$ en tiempo paralelo $O\left(\log^2(n)\right)$.

Desde ahora fijaremos $r \geq 1$. Nuestro algoritmo está basado en el método de Schützenberger-Bertoni.

Sea 1^n una instancia de $\#SAW_r$. Un camino que se auto-evita contenido en la grilla $[n] \times [r]$ puede ser codificado como una cadena de patrones $p_1 ... p_n$ de longitud n.

Sea L_r el lenguaje

$$L_r = \{p_1 p_2..p_n : \psi(p_1 p_2..p_n)\}$$

donde $\psi(p_1 p_2..p_n)$ es la sentencia: la cadena de patrones $p_1 p_2..p_n$ codifica un camino que se auto-evita. Observe que $\#SAW_r(1^n)$ es igual a $f_{L_r}(1^n)$. Si queremos mostrar que $\#SAW_r$ puede ser resuelto en tiempo $O\left(\log^2(n)\right)$ es suficiente mostrar que es posible calcular en tiempo $2^{O(r)}$ un autómata de pila determinístico $\mathcal{M}_r$ tal que

$$f_{L_r}(1^n) = f_{L(\mathcal{M}_r)}(1^n)$$

La prueba del siguiente lema es bastante fácil.

Lema 4.5 *$p_1 p_2..p_n$ pertenece a L_r si y solo si las siguiente condiciones se satisfacen*

1. *Para todo $i \leq n-1$, la pareja (p_i, p_{i+1}) es una pareja admisible.*
2. *$\beta_1(p_1) + \sum_{i \leq n-1} \alpha_1(p_i, p_{i+1}) = 2$ y las ecuaciones $\beta_0(p_1) = \beta_3(p_1) = 0$ se satisfacen.*
3. *La cadena de patrones $p_1 p_2 ... p_n$ evita la creación de ciclos.*

Se sigue del lema 4.5 que el lenguaje L_r puede ser definido como

$$L_r = \{p_1 p_2..p_n : \Psi_1 \wedge \Psi_2\}$$

donde Ψ_1 es la conjunción de las primeras dos condiciones en el enunciado del Lema 4.5 y Ψ_2 es la tercera condición.

Teorema 4.3 *L_r es un lenguaje regular y podemos calcular en tiempo $2^{O(r)}$ un autómata finito $\mathcal{M}_r$ que reconoce L_r.*

Prueba. Recordemos que cualquier lenguaje regular es un lenguaje libre del contexto no ambiguo. Para $1 \leq i \leq 2$ definiremos los conjuntos

$$H_i = \{p_1 ... p_n \in \Sigma^* : \Psi_i\}$$

Observe que $L_r = H_1 \cap H_2$. Recuerde que la intersección de un número finito de lenguajes regulares es también un lenguaje regular. Además, es suficiente mostrar que existen dos autómatas de estado finito que reconocen los lenguajes H_1 y H_2 respectivamente. El Lema 4.4 establece que es posible calcular en tiempo $2^{O(r)}$, ambos autómatas. ■

Observación 4.4 *Es importante señalar que el algoritmo implícito en la prueba del Teorema 4.3 puede ser adaptado fácilmente a cualquier dimensión.*

Bibliografía

[1] Aho, A., Hopcroft, J., and Ullman, J. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Pub. Co., 1974.

[2] Balázs, P. On the number of hv-convex discrete sets. In *Proceedings of the 12th international conference on Combinatorial image analysis* (Berlin, Heidelberg, 2008), IWCIA'08, Springer-Verlag, pp. 112–123.

[3] Bayes, T., Price, R., and Canton, J. *An essay towards solving a problem in the doctrine of chances*. C. Davis, Printer to the Royal Society of London, 1763.

[4] Bertoni, A., Goldwurm, M., and Sabadini, N. The complexity of computing the number of strings of given length in context-free languages. *Theor. Comput. Sci. 86* (September 1991), 325–342.

[5] Bertoni, A., Goldwurm, M., and Santini, M. Random generation and approximate counting of ambiguously described combinatorial structures. *STACS 2000 1770* (2000), 567–580.

[6] Bridson, M. R., and Gilman, R. H. Context-free languages of sub-exponential growth. *J. Comput. Syst. Sci. 64* (March 2002), 308–310.

[7] Chomsky, N., and Schützenberger, M. P. The algebraic theory of context-free languages. In *Computer Programming and Formal Systems*, P. Braffort and D. Hirshberg, Eds., Studies in Logic. North-Holland Publishing, Amsterdam, 1963, pp. 118–161.

[8] Comtet, L. Calcul pratique des coefficients de taylor dúne fonction algébrique. *Enseignement Math 10* (1964), 267–270.

[9] Doi, M., and Edwards, M. *The Theory of Polymer Dynamics*. Oxford University Press, 1986.

[10] Earley, J. An efficient context-free parsing algorithm. *Commun. ACM 26* (January 1983), 57–61.

[11] Flajolet, P., Zimmerman, P., and Van Cutsem, B. A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci. 132* (September 1994), 1–35.

[12] Flory, P. J. *Principles of polymer chemistry*. George Fisher Baker non-resident lectureship in chemistry at Cornell University. Cornell University Press, 1953.

[13] Ginsburg, S., and Spanier, E. H. Bounded algol-like languages. *Transactions of The American Mathematical Society 113* (1964), 333–333.

[14] Hartmanis, J. Context-free languages and turing machine computations. *Proc. in Applied Mathematics, Amer. Math. Soc. 19* (1967), 42–51.

[15] Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*, second ed. Pearson Education, 2001.

[16] Jensen, I. Enumeration of compact self-avoiding walks. *Computer Physics Communications 142*, 1-3 (2001), 109 – 113.

[17] Jerrum, M., Valiant, L. G., and Vazirani, V. V. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci. 43* (1986), 169–188.

[18] Klein, D. J. Asymptotic distributions for self-avoiding walks constrained to strips, cylinders, and tubes. *Journal of Statistical Physics 23* (1980), 561–586. 10.1007/BF01011730.

[19] Montoya, J. A. A note concerning the algorithmic analysis of polymer thermodynamcis. To be published in MATCH, Jul 2011.

[20] Montoya, J. A., Gutierrez, F., and Zambrano, L. E. Some applications of the Schutzenberger-Bertoni method. *Electronic Notes in Discrete Mathematics 37* (2011), 93–98.

[21] Ogihara, M., and Toda, S. The complexity of computing the number of self-avoiding walks in two-dimensional grid graphs and in hypercube graphs. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science* (London, UK, 2001), MFCS '01, Springer-Verlag, pp. 585–597.

[22] Papadimitriou, C. M. *Computational complexity.* Addison-Wesley, 1994.

[23] Randall, D., and Sinclair, A. Testable algorithms for self-avoiding walks. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1994), SODA '94, Society for Industrial and Applied Mathematics, pp. 593–602.

[24] Raz, D. Length considerations in context-free languages. *Theor. Comput. Sci. 183* (August 1997), 21–32.

[25] Reif, J. H. Logarithmic depth circuits for algebraic functions. *SIAM Journal on Computing 15*, 1 (February 1986), 231–242.

[26] Santini, M. *Random Generation and Approximate Counting of Combinatorial Structures.* PhD thesis, Università Degli Studi Di Milano, 2010.

[27] Sipser, M. *Introduction to the Theory of Computation*, second ed. Thomson Course Technology, 2006.

[28] Thompson, C. *Mathematical statistical mechanics.* Series of books in applied mathematics. Macmillan, 1971.

[29] Valiant, L. G. The complexity of computing the permanent. *Theor. Comput. Sci. 8* (1979), 189–201.

[30] Valiant, L. G. The complexity of enumeration and reliability problems. *SIAM Journal on Computing 8*, 3 (1979), 410–421.

[31] Wall, F., and Klein, D. J. Self-avoiding random walks on lattice strips. *Proc. Nat. Acad. Sci. USA 76*, 4 (April 1979), 1529–1531.

[32] Welsh, D. *Complexity: knots, colourings and counting*. London Mathematical Society lecture note series. Cambridge University Press, 1993.

[33] Wich, K., and Kassel, F. M. U. Exponential ambiguity of context-free grammars. In *Proc. DLT* (1999), World Scientific, pp. 125–138.