**Grado en Ingeniería Informática.**

# Portafolio de Prácticas.

---

**Nombre de la asignatura:**
Modelos de Computación.

**Realizado por:**
Néstor Rodríguez Vico

*Este portafolio contiene 7 prácticas (incluidas las dos opcionales), han sido corregidas y puntuadas con la máxima calificación.*


ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS
INFORMÁTICA Y DE TELECOMUNICACIÓN.

---

Granada, 16 de enero de 2017.

# Índice general

# Capítulo 1

# Práctica 1: Gramáticas y lenguajes.

## 1.1. Ejercicio 1.

Sea G = (V, T, P, S), donde V = {S, A, B}, T = {a, b}, el símbolo de partida es S y las reglas son:

| | | | |
|---|---|---|---|
| S→aB | S→bA | A→a | A→aS |
| A→bAA | B→b | B→bS | B→aBB |

Esta gramática genera el lenguaje:

$$L(G) = \{u \mid u \in \{a, b\}^+ \text{ y } N_a(u) = N_b(u)\}$$

donde $N_a(u)$ y $N_b(u)$ son el número de apariciones de símbolos a y b en u, respectivamente.

Para demostrar que esta gramática genera dicho lenguaje tenemos que demostrar dos ideas:
- Todas las palabra generadas son del tipo $N_a(u) = N_b(u)$.
- Que se generan todas las cadenas posibles con $N_a(u) = N_b(u)$.

Pasemos a demostrar la primera de ellas. Para ello usaremos las siguientes interpretaciones:
- A: palabra con una $a$ de más.
- B: palabra con una $b$ de más.
- S: palabra con igual número de $a$ que $b$.

Las 3 producciones que generamos tirando de B mantienen la coherencia de

nuestra interpretación. Dichas producciones son: B→b, B→bS (dado que en S se da $N_a(u) = N_b(u)$ y se le añade la $b$ previa) y B→aBB (dado que la primera B anula la $a$ que se añade al principio y la segunda B concede la $b$ extra que se necesita).
Al igual sucede paras las producciones que generamos tirando de A y de S.
De esta manera queda demostrado que mediante dicha gramática sólo se generan palabras con $N_a(u) = N_b(u)$.

A continuación vamos a demostrar la segunda idea. Podemos generar cadenas que empiezan por $a$ usando la regla S→aB y cadenas que empiecen por $b$ usando la regla S→bA. Elegimos una opción de las dos anteriores, por ejemplo, generar cadenas que empiecen por $b$. ¿Es posible que el segundo elemento de la cadena sea una $a$? Sí, usando la $4^a$ regla, A→aS, así tenemos S→bA→baS. ¿En lugar de una $a$ podría ir una $b$? Sí, usando la $5^a$ regla, A→bAA, así tenemos S→bA→bbAA. Al igual sucedería si en el primer paso hubiésemos elegido empezar por $a$ en vez de por $b$.

Este proceso se puede repetir indefinidamente generando así todas las cadenas posibles.

## 1.2. Ejercicio 2.

Sea G = ({S, X, Y}, {a, b, c}, P, S) donde P tiene las reglas:

| | | | |
|---|---|---|---|
| S→abc | S→aXbc | Xb→bX | Xc→Ybcc |
| bY→Yb | aY→aaX | aY→aa | |

Esta gramática genera el lenguaje:

$$\{a^n b^n c^n \mid n = 1, 2, ...\}$$

Al igual que en el ejercicio anterior, la idea es demostrar que la gramática G genera el lenguaje L. Inicialmente tenemos dos posibilidades: S→abc y S→aXbc. ¿Que hago para generar la cadenas *aabbcc*? Partimos de la regla S→aXbc. A continuación usamos la regla Xb→bX y así tenemos *abXc*. Luego usamos Xc→Ybcc para conseguir *aYbbcc*. Finalmente usamos la regla aY→aa para obtener la cadena deseada, *aabbcc*.

Podemos observar que la variable $X$ es una variable de desplazamiento hacia la derecha, buscando el elemento $c$ y que la variable $Y$ es una variable de desplazamiento hacia la izquierda, buscando el elemento $a$.

Este proceso se puede repetir de forma indefinida para ir aumentando el número de $a$, $b$ y $c$.

# Capítulo 2

# Práctica 2: Gramáticas regulares.

Determinar si la gramática G = ({S, A, B}, {a, b, c, d}, P, S) donde P es el conjunto de reglas de producción:

$$S \rightarrow AB \quad A \rightarrow Ab \quad A \rightarrow a \quad B \rightarrow cB \quad B \rightarrow d$$

genera un lenguaje de tipo 3.

## 2.1. Solución.

Lo primero que debemos hacer es ver que lenguaje genera dicha gramática. Para ver esto lo que debemos hacer es partir del símbolo inicial y aplicar reglas de producción de manera "aleatoria" hasta poder identificar un patrón que nos indique cual es el lenguaje que genera dicha gramática.

Si partimos de S→AB y aplicamos la regla de producción A→Ab un número de veces, $i$, obtendremos una cadena del estilo $Ab..bB$ con $i$ número de $b$. Si a continuación aplicamos la regla B→cB un número de veces, $j$, obtendremos una cadena del estilo $Ab..bc..cB$ con $j$ número de $c$. Si aplicamos estas dos reglas cualquier número de veces, obtendremos una cadena cada vez más larga con un mayor número de símbolos terminales $a$ y $b$. Para terminar de generar una cadena debemos sustituir las variables restantes por símbolos terminales usando las reglas de producción disponibles, en este caso podemos cambiar la variable $A$ por el símbolo terminal $a$ usando la regla de producción A→a y la variable $B$ por el símbolo terminal $d$ usando la regla de producción B→d. De esta manera nuestra cadena generada tendría el estilo

*ab..bc..cd.*

Una vez hemos visto la cadena generada, podemos darnos observar que el lenguaje generado es el siguiente:

$$\mathrm{L} = \{\mathrm{ab}^i\mathrm{c}^j\mathrm{d} : \mathrm{i, j} \in \mathrm{N}\}$$

Podemos ver que no se trata de una gramática de tipo 3 observando las reglas de producción de la gramática. ¿Por qué no es de tipo 3? Para que sea una gramática de tipo 3, todas las reglas de producción deben tener el formato:

$$\mathrm{A}\rightarrow\mathrm{uB} \text{ ó } \mathrm{A}\rightarrow\mathrm{u}$$

es decir, en la parte izquierda de la "flecha" sólo puede haber una variable y en la parte derecha un conjunto de símbolos terminales y al final de todo, opcionalmente, una única variable. Esta condición no la cumple la regla de producción S→AB, por lo tanto no se trata de una gramática tipo 3.

Pero, ¿se podría generar este mismo lenguaje usando una gramática de tipo 3? Para ello habría que cambiar las reglas de producción por otras.

Como queremos obtener una cadena que empiece por $a$ y continué por un número $i$ de $b$ lo más lógico sería partir de la regla de producción S→aB y a continuación aplicar la misma regla que la proporcionada en el enunciado del ejercicio, B→bB, de esta manera podemos añadir el número que deseemos de $b$.Con estas dos reglas podemos obtener una cadena del estilo *ab..bB*. A continuación nos interesa generar una sucesión finita de $c$, por lo que sería lógico pensar añadir la regla de producción B→cB. Así de podría añadir cualquier número de $c$. Esta regla de producción genera la cadena que estamos deseando, pero, ¿sólo esa cadena? La respuesta es no, ya que si aplicamos de manera intercalada las reglas de producción B→bB y B→cB obtendríamos una cadena del estilo *ab..bc..cb..bc..* y así sucesivamente. Para corregir esto, lo que debemos hacer es cambiar la regla anterior por C→cC y añadir una regla que nos permita cambiar la variable $B$ por la variable $C$, dicha regla es B→C. De esta manera, aplicando repetidas veces las reglas de producción, tendríamos una cadena del estilo *ab..bc..cD*. Finalmente y para terminar, añadimos la regla C→d para obtener la cadena que estamos deseando.

Las reglas de producción de nuestra nueva gramática son:

$$\mathrm{S}\rightarrow\mathrm{aB} \quad \mathrm{B}\rightarrow\mathrm{bB} \quad \mathrm{B}\rightarrow\mathrm{C} \quad \mathrm{C}\rightarrow\mathrm{cC} \quad \mathrm{C}\rightarrow\mathrm{d}$$

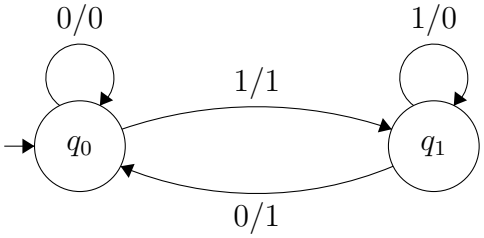
# Capítulo 3

# Práctica 3: Máquinas de estados.

En esta práctica vamos a desarrollar una máquina de estados que se encarga de cifrar y descifrar cadenas dadas en la cinta de entrada.

## 3.1. Cifrado.

La máquina de estados que se encarga de cifrar la cadena es la siguiente:




Esta máquina de estados está construida a partir de la siguiente tabla:

| **Estado** | **Símbolo anterior leído** | **Símbolo leído** | **Símbolo escrito** |
|---|---|---|---|
| $q_o$ | Primer símbolo | 0 | 0 |
| | | 1 | 1 |
| | 0 | 0 | 0 |
| | | 1 | 1 |
| $q_1$ | 1 | 0 | 1 |
| | | 1 | 0 |

Usaremos la siguiente interpretación:

- $q_0$*:* el símbolo anterior leído es 0 o es el primer símbolo que se lee.
- $q_1$*:* el símbolo anterior leído es 1.

Los estados se usan para "recordar" el valor que se ha leído anteriormente. Por ejemplo, si vamos o nos quedamos en $q_0$, "recordamos" que hemos leído un 0.

Para la cadena de entrada *1000101* se obtiene la cadena cifrada *1100111*.

## 3.2. Descifrado.

Al igual que para la encriptación había que fijarse en la entrada de la máquina de estados, en este caso tenemos que fijarnos en la salida. La idea es similar, sólo que en vez de escribir 0 ó 1 en función del número leído, lo hacemos en función del número escrito con anterioridad.

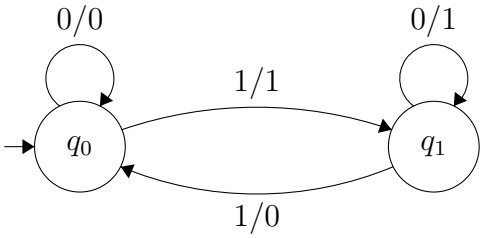
Para ver que elemento debemos escribir en la cinta de salida, usaremos la siguiente tabla:

| **Estado** | **Símbolo anterior escrito** | **Símbolo escrito** | **Símbolo escrito** |
|---|---|---|---|
| $q_o$ | Primer símbolo | 0 | 0 |
| | | 1 | 1 |
| | 0 | 0 | 0 |
| | | 1 | 1 |
| $q_1$ | 1 | 0 | 1 |
| | | 1 | 0 |

De esta manera, para la cinta de entrada cifrada *1100111* obtendríamos la salida *1000101*.

Finalmente, para ver la dirección de las transiciones debemos ver lo que se ha escrito, es decir, lo que está a la derecha de la barra en cada estado.

Con estas pautas, la máquina de estados que obtenemos es la siguiente:


0/0
0/1
1/1
$q_0$
$q_1$
1/0

# Capítulo 4

# Práctica 4: Autómatas con lex.

## 4.1. Parte 1.

En esta práctica vamos a manejar ficheros lex. El autómata que voy a usar es el que hemos visto en clase de teoría. El fichero lex es el siguiente:

```
car         [a-zA-Z]
digito      [0-9]
signo       (\-|\+)
suc         ({digito}+)
enter       ({signo}?{suc})
real1       ({enter}\.{digito}*)
real2       ({signo}?\.{suc})

            int ent=0, real=0, ident=0, sumaent=0;
%%
            int i;
{enter}                         {ent++; sscanf(yytext," %d",&
    ↪ i); sumaent += i; printf("Numero entero %s\n",
    ↪ yytext);}
({real1}|{real2})               {real++; printf("Num. real %s
    ↪ \n",yytext);}
{car}({car}|{digito})*          {ident++; printf("Var. ident.
    ↪  %s\n",yytext);}
.|\n                            {;}
%%
yywrap()                        {printf("Numero de Enteros %d
    ↪ , reales %d, ident %d, Suma de Enteros %d",ent,
    ↪ real,ident,sumaent); return 1;}
```

Una vez tenemos el fichero lex preparado, preparamos el fichero de entrada con el que trabajará nuestro autómata. En mi caso el dichero de entrada es el siguiente:

```
3.3
+3
-3
-33333333
597.89
-3
0
ModelosDeComputacion
-303030
```

Para ver que funciona correctamente tenemos que seguir los siguientes pasos:

1. Ejecutamos lex sobre nuestro fichero *p4.lex*: *lex p4.lex*
2. Compilamos el programa que ha creado lex: *gcc lex.yy.c -o p4 -ll*
3. Ejecutamos *p4* pasándole como entrada nuestro fichero de entrada, *entrada*, y redirigiendo la salida a un fichero, *salida*, para ver el resultado obtenido: *p4 < entrada > salida*

Una vez hemos seguido los pasos, el contenido del fichero *salida* es el siguiente:

```
Num. real 3.3
Numero entero +3
Numero entero -3
Numero entero -33333333
Num. real 597.89
Numero entero -3
Numero entero 0
Var. ident. ModelosDeComputacion
Numero entero -303030
Numero de Enteros 6, reales 2, ident 1, Suma de Enteros
    ↪ -33636366
```

## 4.2. Wordcount con *lex*.

Esta parte de la práctica la he realizado con Míriam Mengíbar Rodríguez. Hemos hecho un programa en C apoyado en *lex* que implementa la utilidad *wordcount* disponible en los sistemas UNIX. Primero hemos definido 3 variables globales para llevar cuenta de lo procesado hasta el momento. También hemos definido cuatro términos a reconocer: *caracter*, *digito*, *palabra* y *linea*. Finalmente, hemos definido tres reglas una para contar el número de líneas,

otra para contar el número de palabras y otra para contar el número de caracteres. El programa obtenido es el siguiente:

```
%{
#include <stdio.h>
#include <stdlib.h>
int num_caracteres = 0, num_palabras = 0, num_lineas = 0;
%}

caracter [a-zA-ZáäàâÁÄÀÂéëêèÉËÈÊíïìîÍÏÌÎóöòôÓÖÒÔúùûüÚÜÙûçÇñÑ]
digito [0-9]
palabra ({caracter}|{digito})+
linea \n

%%

{linea} {num_caracteres++; num_lineas++;}
{palabra} {num_palabras++; num_caracteres += strlen(yytext);}
. {num_caracteres++;}

%%

int main(){
 yylex();
 printf("Numero de caracteres: %d // ", num_caracteres);
 printf("Numero de palabras: %d // ", num_palabras);
 printf("Numero de lineas: %d\n", num_lineas);
}
```

Para ver que funciona correctamente tenemos que seguir los siguientes pasos:

1. Ejecutamos lex sobre nuestro fichero *wordcount.l*: *lex wordcount.l*
2. Compilamos el programa que ha creado lex: *gcc lex.yy.c -o wc -ll*
3. Ejecutamos *wc* pasándole como entrada nuestro fichero de entrada, *entrada*, y redirigiendo la salida a un fichero, *salida*, para ver el resultado obtenido: *wc <entrada> salida*

Como fichero de entrada voy a usar un poema de Federico García Lorca. El poema es el siguiente:

Todos te desean pero ninguno te ama.
Nadie puede quererte, serpiente,
porque no tienes amor,
porque estás seca como la paja seca

y no das fruto.
Tienes el alma como la piel de los viejos.
Resígnate. No puedes hacer más
sino encender las manos de los hombres
y seducirlos con las promesas de tu cuerpo.
Alégrate. En esa profesión del deseo
nadie como tú para simular inocencia
y para hechizar con tus ojos inmensos.

El resultado de la ejecución de nuestro fichero lex sobre dicho poema es el siguiente:

```
Numero de caracteres: 432 // Numero de palabras: 74 // Numero de
    ↪ lineas: 12
```

# Capítulo 5

# Práctica 5: Gramáticas ambiguas.

Sea la gramática:

$$E \rightarrow I \quad E \rightarrow I\text{-}E \quad E \rightarrow E\text{-}I \quad I \rightarrow a|b|c|d$$

¿Es posible encontrar una gramática que genere el mismo lenguaje pero que no sea ambigua y por lo tanto el lenguaje no sea inherentemente ambiguo?

## 5.1. Solución.

La respuesta es que sí, simplemente basta con eliminar la regla de producción E→I-E o la regla de producción E→E-I. Por lo tanto, podemos tener dos gramáticas que generan dicho lenguaje y no son ambiguas. La primera de ellas sería:

$$E \rightarrow I \quad E \rightarrow E\text{-}I \quad I \rightarrow a|b|c|d$$

La segunda gramática sería:

$$E \rightarrow I \quad E \rightarrow I\text{-}E \quad I \rightarrow a|b|c|d$$

# Capítulo 6

# Práctica 6: Forma normal de *Chomsky.*

## 6.1. Enunciado.

Sea la gramática $G = (\{S, A, B, C, D, E, F\}, \{a, b, c\}, P_0, S)$ y $P_0$:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→B | A→aCC | A→baD |
| B→cBD | B→$\varepsilon$ | B→AC | C→bD | C→aBA | D→CD |
| D→a | D→EF | E→Eb | F→a | | |

Obtenga la gramática correspondiente a la gramática $G$ en forma normal de *Chomsky*.

## 6.2. Solución.

Para llegar a la forma normal de *Chomsky* vamos a seguir los siguientes pasos:

1. Eliminar símbolos y producciones inútiles.
2. Eliminar producciones nulas.
3. Eliminar producciones unitarias.
4. Normalizar la gramática en forma normal de *Chomsky*.

### 6.2.1. Eliminar símbolos y producciones inútiles.

Para este paso, vamos a usar una estructura adicional, $V_t$ en la cual vamos a ir introduciendo las variables que se encuentran a la izquierda de reglas de producción que a la derecha tienen un símbolo terminal. Tras introducir dichas variables en $V_t$, realizamos sucesivas pasadas introduciendo las variables que se encuentran a la izquierda de reglas de producción que a la derecha tienen una o varias variables que ya están en $V_t$. El proceso termina cuando tras una pasada por la gramática no se añade ninguna variable a $V_t$. El proceso lo podemos ver a continuación:

Pasada número 1. $V_t = \{\}$
Pasada número 2. $V_t = \{S, D, F\}$
Pasada número 3. $V_t = \{S, D, F, A, C\}$
Pasada número 4. $V_t = \{S, D, F, A, C, B\}$
Pasada número 5. $V_t = \{S, D, F, A, C, B\}$ $\rightarrow$ Fin del proceso.

A continuación, eliminamos las variables que no están en $V_t$ y las reglas de producción que emplean dichas variables. $V$ - $V_t = \{E\}$. Una vez eliminadas dichas producciones, la gramática que obtenemos es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S$\rightarrow$bDD | S$\rightarrow$Ca | S$\rightarrow$bc | A$\rightarrow$B | A$\rightarrow$aCC | A$\rightarrow$baD |
| B$\rightarrow$cBD | B$\rightarrow\varepsilon$ | B$\rightarrow$AC | C$\rightarrow$bD | C$\rightarrow$aBA | D$\rightarrow$CD |
| D$\rightarrow$a | F$\rightarrow$a | | | | |

A continuación, realizamos una nueva pasada sobre la gramática, esta vez usando tres estructuras: $V_s$ para almacenar las variables ya procesadas, $J$ para almacenar las variables pendientes de procesar y $T_s$ para almacenar los símbolos terminales generables desde variables pertenecientes a $V_s$. Los pasos a seguir serían:

1. Sacamos una variable de $J$.
2. Introducimos en $J$ las variables que se encuentran a la derecha en reglas de producción donde dicha variable se encuentra a la izquierda.
3. Introducimos en $T_s$ los símbolos terminales que se encuentran a la derecha en reglas de producción donde dicha variable se encuentra a la izquierda.
4. Introducimos la variable en $V_s$.

Este proceso se realiza hasta que no haya ninguna variable pendiente de procesar, es decir, $J$ esté vacío. El proceso lo podemos ver a continuación:

| | | | |
|---|---|---|---|
| Pasada número 1. | $V_s = \{\}$ | J = {S} | $T_s = \{\}$ |
| Pasada número 2. | $V_s = \{S\}$ | J = {D, C} | $T_s = \{a, b, c\}$ |
| Pasada número 3. | $V_s = \{S, D\}$ | J = {C} | $T_s = \{a, b, c\}$ |
| Pasada número 4. | $V_s = \{S, D, C\}$ | J = {B, A} | $T_s = \{a, b, c\}$ |
| Pasada número 5. | $V_s = \{S, D, C, B\}$ | J = {A} | $T_s = \{a, b, c\}$ |
| Pasada número 6. | $V_s = \{S, D, C, B, A\}$ | J = {} | $T_s = \{a, b, c\}$ |

A continuación, eliminamos las variables que no están en $V_s$ y las reglas de producción que emplean dichas variables. $V$ - $V_s = \{F\}$. Una vez eliminadas dichas producciones, la gramática que obtenemos es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→B | A→aCC | A→baD |
| B→cBD | B→$\varepsilon$ | B→AC | C→bD | C→aBA | D→CD |
| D→a | | | | | |

### 6.2.2. Eliminar producciones nulas.

Partimos de la gramática obtenida en el paso anterior. Para eliminar las producciones nulas hay que realizar dos pasos; primero identificar las variables anulables de forma directa (A→$\varepsilon$) y las variables cuyas producciones tengan todas las variables de la derecha anulables. Estas variables las guardamos en la estructura adicional $H$. Este proceso lo realizamos de manera reiterada hasta que, tras una pasada por la gramática, no se añade ninguna variable a $H$. El proceso lo podemos ver a continuación:

| | | |
|---|---|---|
| Pasada número 1. | H = {} | |
| Pasada número 2. | H = {B} | |
| Pasada número 3. | H = {B, A} | |
| Pasada número 4. | H = {B, A} | →Fin del proceso. |

Una vez acabado el proceso, eliminamos las reglas de producción que tienen a la izquierda variables que son directamente anulables. La gramática que obtenemos es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→B | A→aCC | A→baD |
| B→cBD | B→AC | C→bD | C→aBA | D→CD | D→a |

En el segundo paso, debemos añadir las producciones necesarias para seguir generando el mismo lenguaje que con la gramática que contenía producciones nulas. La idea es analizar las producciones que tienen a la derecha variables que están en $H$ y añadir dicha regla de producción pero modificada para permitir generar el mismo lenguaje. Las reglas a añadir las podemos ver a continuación:

- $A \rightarrow B$ no genera ninguna regla nueva.
- $B \rightarrow cBD$ genera $B \rightarrow cD$.
- $B \rightarrow AC$ genera $B \rightarrow C$.
- $C \rightarrow aBA$ genera tres reglas; $C \rightarrow aB$, $C \rightarrow aA$ y $C \rightarrow a$.

Una vez añadimos dichas reglas de producción, la gramática obtenida es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→B | A→aCC | A→baD |
| B→cBD | B→AC | C→bD | C→aBA | D→CD | D→a |
| B→cD | B→C | C→aB | C→aA | C→a | |

### 6.2.3. Eliminar producciones unitarias.

Las producciones unitarias son de la forma A→B. Lo primero que debemos hacer es añadir a la lista $H$ las parejas de variables que forman producciones unitarias.

$$H = \{(A, B), (B, C)\}$$

A continuación, buscamos en $H$ dos parejas tal que la segunda variable de la primera pareja sea igual a la primera variable de la segunda pareja y añadimos la primera variable de la primera pareja y la segunda variable de la segunda pareja como una nueva pareja. En este caso, añadiríamos la pareja (A, C).

$$H = \{(A, B), (B, C), (A, C)\}$$

Una vez tenemos $H$ construido, eliminamos las producciones unitarias. La gramática generada es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→aCC | A→baD | B→cBD |
| B→AC | C→bD | C→aBA | D→CD | D→a | B→cD |
| C→aB | C→aA | C→a | | | |

Finalmente, como debemos añadir las reglas necesarias para seguir produciendo el mismo lenguaje tras eliminar las producciones unitarias. Para ello cogemos las parejas de $H$ y las producciones que tienen a la izquierda la segunda variable de la pereja. A continuación, en la regla de producción, cambiamos la variable de la izquierda por la primera variable de la pareja elegida y añadimos esta nueva regla. Este proceso lo podemos ver de manera más clara a continuación:

| | | |
|---|---|---|
| (A, B) | B→cBD | A→cBD |
| | B→AC | A→AC |
| | B→cD | A→cD |
| (B, C) | C→bD | B→bD |
| | C→aBA | B→aBA |
| | C→aB | B→aB |
| | C→aA | B→aA |
| (A, C) | C→bD | A→bD |
| | C→aBA | A→aBA |
| | C→aB | A→aB |
| | C→aA | A→aA |

La gramática obtenida es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| S→bDD | S→Ca | S→bc | A→aCC | A→baD | B→cBD |
| B→AC | C→bD | C→aBA | D→CD | D→a | B→cD |
| C→aB | C→aA | C→a | A→cBD | A→AC | A→cD |
| B→bD | B→aBA | B→aB | B→aA | A→bD | A→aBA |
| B→aB | B→aA | | | | |

### 6.2.4. Normalizar la gramática en forma normal de *Chomsky*.

Las producciones de una gramática en forma normal de *Chomsky* son del estilo A→BC o A→a. Lo primero que hacemos, es añadir una variable para cada símbolo terminal, su regla de producción correspondiente y cambiamos la aparición de los símbolos terminales por las nuevas variables en todas las reglas de producción. La nueva gramática es la siguiente:

| | | | | | |
|---|---|---|---|---|---|
| $S \to T_b DD$ | $S \to CT_a$ | $S \to T_b T_c$ | $A \to T_a CC$ | $A \to T_b T_a D$ | $B \to T_c BD$ |
| $B \to AC$ | $C \to T_b D$ | $C \to T_a BA$ | $D \to CD$ | $D \to T_a$ | $B \to T_c D$ |
| $C \to T_a B$ | $C \to T_a A$ | $C \to T_a$ | $A \to T_c BD$ | $A \to AC$ | $A \to T_c D$ |
| $B \to T_b D$ | $B \to T_a BA$ | $B \to T_a B$ | $B \to T_a A$ | $A \to T_b D$ | $A \to T_a BA$ |
| $B \to T_a B$ | $B \to T_a A$ | $T_a \to a$ | $T_b \to b$ | $T_c \to c$ | |

A continuación, debemos eliminar las producciones que no son del estilo $A \to BC$ o $A \to a$ y añadimos las reglas necesarias para poder generar el mismo lenguaje. Las reglas a modificar son las siguientes:

1. $S \to T_b DD$ la cambiamos por $S \to T_b X_1$ y $X_1 \to DD$.
2. $A \to T_a CC$ la cambiamos por $A \to T_a X_2$ y $X_2 \to CC$.
3. $A \to T_b T_a D$ la cambiamos por $A \to T_b X_3$ y $X_3 \to T_a D$.
4. $B \to T_c BD$ la cambiamos por $B \to T_c X_4$ y $X_4 \to BD$.
5. $C \to T_a BA$ la cambiamos por $C \to T_a X_5$ y $X_5 \to BA$.
6. $A \to T_c BD$ la cambiamos por $A \to T_c X_4$.
7. $B \to T_a BA$ la cambiamos por $B \to T_a X_5$.
8. $A \to T_a BA$ la cambiamos por $A \to T_a X_5$.

Finalmente, la gramática original en forma normal de *Chomsky* es la que podemos ver a continuación:

| | | | | | |
|---|---|---|---|---|---|
| $S \to CT_a$ | $S \to T_b T_c$ | $B \to AC$ | $C \to T_b D$ | $D \to CD$ | $D \to T_a$ |
| $B \to T_c D$ | $C \to T_a B$ | $C \to T_a A$ | $C \to T_a$ | $A \to AC$ | $A \to T_c D$ |
| $B \to T_b D$ | $B \to T_a B$ | $B \to T_a A$ | $A \to T_b D$ | $B \to T_a B$ | $B \to T_a A$ |
| $T_a \to a$ | $T_b \to b$ | $T_c \to c$ | $S \to T_b X_1$ | $X_1 \to DD$ | $A \to T_a X_2$ |
| $X_2 \to CC$ | $A \to T_b X_3$ | $X_3 \to T_a D$ | $B \to T_c X_4$ | $C \to T_a X_5$ | $X_5 \to BA$ |
| $A \to T_c X_4$ | $B \to T_a X_5$ | $A \to T_a X_5$ | | | |

# Capítulo 7

# Práctica 7: Obtener una gramática a partir de un autómata con pila.

## 7.1. Enunciado.

Obtener la gramática que genera las cadenas que acepta el autómata que podemos ver a continuación. El autómata acepta cadenas que tienen igual número de 0 que de 1. El autómata es el siguiente:

$$M = (\{q_1\}, \{0,1\}, \{R, X, Y\}, \delta, q_1, R, \emptyset)$$

$$\begin{array}{ll} \delta(q_1, 0, R) = \{(q_1, XR)\} & \delta(q_1, 0, X) = \{(q_1, XX)\} \\ \delta(q_1, 1, R) = \{(q_1, YR)\} & \delta(q_1, 1, Y) = \{(q_1, YY)\} \\ \delta(q_1, 1, X) = \{(q_1, \epsilon)\} & \delta(q_1, 0, Y) = \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, R) = \{(q_1, \epsilon)\} & \end{array}$$

## 7.2. Solución.

Lo primero que hacemos es añadir una nueva regla por cada estado. Esta regla sería $S \to [q_1, Z_0, q_1]$. A continuación añadimos las reglas que simulan las transiciones del autómata que no meten nada en la pila. Estás reglas son:

- Para $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$ añadimos $[q_1, X, q_1] \to 1$
- Para $\delta(q_1, 0, Y) = \{(q_1, \epsilon)\}$ añadimos $[q_1, Y, q_1] \to 0$
- Para $\delta(q_1, \epsilon, R) = \{(q_1, \epsilon)\}$ añadimos $[q_1, R, q_1] \to \epsilon$

Finalmente añadimos las reglas de producción que simulan las transiciones del autómata que si introducen elementos en la pila. Estas reglas son:

- Para $\delta(q_1, 0, R) = \{(q_1, XR)\}$ añadimos $[q_1, R, q_1] \to 0[q_1, X, q_1][q_1, R, q_1]$
- Para $\delta(q_1, 0, X) = \{(q_1, XX)\}$ añadimos $[q_1, X, q_1] \to 0[q_1, X, q_1][q_1, X, q_1]$
- Para $\delta(q_1, 1, R) = \{(q_1, YR)\}$ añadimos $[q_1, R, q_1] \to 1[q_1, Y, q_1][q_1, R, q_1]$
- Para $\delta(q_1, 1, Y) = \{(q_1, YY)\}$ añadimos $[q_1, Y, q_1] \to 1[q_1, Y, q_1][q_1, Y, q_1]$

La gramática obtenida es la siguiente:

$$
\begin{array}{ll}
[q_1, R, q_1] \to 0[q_1, X, q_1][q_1, R, q_1] & [q_1, X, q_1] \to 0[q_1, X, q_1][q_1, X, q_1] \\
[q_1, R, q_1] \to 1[q_1, Y, q_1][q_1, R, q_1] & [q_1, Y, q_1] \to 1[q_1, Y, q_1][q_1, Y, q_1] \\
[q_1, X, q_1] \to 1 & [q_1, Y, q_1] \to 0 \\
[q_1, R, q_1] \to \varepsilon &
\end{array}
$$