

## Grado en Ingeniería Informática

### Computabilidad y Algoritmia

---

# PRÁCTICA 7

## Gramáticas en Forma Normal de Chomsky

Factor de ponderación: 10

Semana del 30 de octubre al 3 de noviembre de 2023

### 1. Objetivos

El objetivo de la práctica es consolidar los conocimientos adquiridos sobre Gramáticas [1] al mismo tiempo que se continúan desarrollando capacidades para diseñar y desarrollar programas orientados a objetos en C++. El programa leerá desde un fichero la especificación de una gramática independiente del contexto y a partir de dicha definición generará automáticamente una gramática equivalente en Forma Normal de Chomsky.

Podemos entonces concretar que los objetivos principales de esta práctica son los siguientes:

- Consolidar los conocimientos adquiridos sobre gramáticas.
- Identificar tipos de gramáticas en función de sus producciones.
- Estudiar y practicar el algoritmo de transformación de una gramática a su forma normal de Chomsky.
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

## 2. Gramáticas independientes del contexto

Desde la antigüedad, los lingüistas han descrito las gramáticas de los idiomas en términos de su estructura de bloques, y han descrito cómo las oraciones se construyen recursivamente a partir de frases más pequeñas y, finalmente, de palabras o elementos de palabras individuales. Una propiedad esencial de estas estructuras de bloques es que las unidades lógicas nunca se solapan. Por ejemplo, la frase en inglés:

*John, whose blue car was in the garage, walked to the grocery store*

usando corchetes como meta-símbolos puede agruparse como:

[ John [, [ whose [ blue car ] ] [ was [ in [ the garage ] ] ], ] [ walked [ to [ the [ grocery store ] ] ] ]

Una gramática independiente del contexto (*Context Free Grammar, CFG*) [2] proporciona un mecanismo simple y matemáticamente preciso para describir los métodos por los cuales las cadenas de algún lenguaje formal se construyen a partir de bloques más pequeños, capturando la “estructura de bloques” de las frases de manera natural. Su simplicidad hace que este formalismo sea adecuado para un estudio matemático riguroso. En cualquier caso, las características de la sintaxis de los lenguajes naturales no se pueden modelar mediante gramáticas independientes del contexto.

Este formalismo y también su clasificación como un tipo especial de gramática formal fue desarrollado a mediados de los años 50 del siglo pasado por Noam Chomsky [7]. Basándose en esta notación de gramáticas basada en reglas, Chomsky agrupó los lenguajes formales en una serie de cuatro subconjuntos anidados conocidos como la *Clasificación de Chomsky* [8]. Esta clasificación fue y sigue siendo fundamental para la teoría de lenguajes formales, especialmente para la teoría de los lenguajes de programación y el desarrollo de compiladores.

La estructura de bloques fue introducida en los lenguajes de programación por el proyecto Algol (1957-1960), el cual, como consecuencia, también incluía una gramática independiente del contexto para describir la sintaxis del lenguaje. Esto se convirtió en una característica estándar de los lenguajes de programación, aunque no solo de éstos. En [9], a modo de ejemplo, se utiliza una gramática independiente del contexto para especificar la sintaxis de JavaScript.

Las gramáticas independientes del contexto son lo suficientemente simples como para permitir la construcción de algoritmos de análisis eficientes que, para una cadena dada, determinan si - y cómo - se puede generar esa cadena a partir de la gramática. El *algoritmo de Cocke-Younger-Kasami* es un ejemplo de ello, mientras que los ampliamente usados analizadores LR y LL son algoritmos más simples que tratan sólo con subconjuntos más restrictivos de gramáticas independientes del contexto.

Formalmente una *gramática independiente del contexto*  $G$  (por simplicidad nos referiremos a “una gramática”) viene definida por una tupla  $G \equiv (\Sigma, V, S, P)$  cada uno de cuyos componentes se explican a continuación:

- $\Sigma$ : Conjunto de símbolos terminales (o alfabeto de la gramática),  $\Sigma \cap V = \emptyset$
- $V$ : Conjunto de símbolos no terminales ( $V \neq \emptyset$ )
- $S$ : Símbolo de arranque (*S*start) o axioma de la gramática ( $S \in V$ )
- $P$ : Conjunto de reglas de producción:

$$P = \{A \rightarrow \alpha \mid A \in V, \alpha \in (V \cup \Sigma)^*\}$$

$$P \subset V \times (V \cup \Sigma)^*, (P \neq \emptyset)$$

Habitualmente, para especificar una gramática, se especifican todas sus reglas de producción ( $P$ ), y se sigue un convenio de notación que permite determinar todos los elementos. El símbolo de arranque es aquel cuyas producciones aparecen en primer lugar en la lista de producciones. Por ejemplo, las siguientes producciones:

$$S \rightarrow U \mid W$$

$$U \rightarrow TaU \mid TaT$$

$$W \rightarrow TbW \mid TbT$$

$$T \rightarrow aTbT \mid bTaT \mid \epsilon$$

definen una gramática independiente del contexto para el lenguaje de cadenas de letras  $a$  y  $b$  en las que hay un número diferente de unas que de otras. Como se puede apreciar, a partir de las producciones anteriores podremos deducir la especificación completa de la gramática:

- $\Sigma = \{a, b\}$
- $V = \{S, U, W, T\}$
- El símbolo de arranque es  $S$
- Las reglas de producción son las de la relación anterior

### 3. Forma Normal de Chomsky

En teoría de lenguajes formales se dice que una gramática independiente del contexto  $G \equiv (V, \Sigma, S, P)$  está en forma normal de Chomsky (en honor a quien propuso este formato, CNF, *Chomsky Normal Form*) si:

- $V$  sólo contiene símbolos útiles.
- Todas las producciones de  $G$  tienen una de las formas:  

$$A \rightarrow BC$$

$$A \rightarrow a$$

Si  $\epsilon \in L(G)$  se permite además una única producción  $S \rightarrow \epsilon$  y en este caso no se permite que el símbolo de arranque figure en la parte derecha de ninguna regla de producción.

Al margen del interés teórico que tiene la forma normal de Chomsky (se utiliza por ejemplo en la demostración del lema del bombeo para lenguajes independientes del contexto), la conversión a CNF se utiliza como paso previo de otros algoritmos, como el algoritmo de análisis sintáctico (algoritmo de Cocke, Younger y Kasami, CYK [4]).

Cualquier gramática independiente del contexto puede ser transformada en una equivalente escrita en forma normal de Chomsky. Por otra parte, se puede demostrar que en una Gramática escrita en CNF, la derivación de una cadena de  $n$  símbolos tiene exactamente  $2n - 1$  pasos de derivación. Esta propiedad permite determinar si una cadena  $w \in L(G)$  mediante una inspección exhaustiva de todas las derivaciones.

El Algoritmo 1 aplicado a una gramática independiente del contexto permite su transformación a forma normal de Chomsky.

---

**Algorithm 1** Transformación de una CFG a su Forma Normal de Chomsky
 

---

```

1: for all ( $A \rightarrow X_1X_2 \dots X_n$  (con  $n \geq 2$ ,  $X_i \in (\Sigma \cup V)$ ) do
2:   for all ( $X_i$ ) do
3:     if ( $X_i = a \in \Sigma$ ) then
4:       Add the production  $C_a \rightarrow a$ ;
5:       Replace  $X_i$  with  $C_a$  in  $A \rightarrow X_1X_2 \dots X_n$ ;
6:     end if
7:   end for
8: end for
9: for all ( $A \rightarrow B_1B_2 \dots B_m$  (con  $m \geq 3$ ,  $B_i \in V$ ) do
10:  Add  $m - 2$  non-terminal symbols  $D_1D_2 \dots D_{m-2}$ ;
11:  Replace the production  $A \rightarrow B_1B_2 \dots B_m$  with productions:
12:     $A \rightarrow B_1D_1$ 
13:     $D_1 \rightarrow B_2D_2$ 
14:    ...
15:     $D_{m-2} \rightarrow B_{m-1}B_m$ 
16: end for

```

---

Es importante tener en cuenta que **el algoritmo toma como entrada una gramática que no contenga símbolos ni producciones inútiles, unitarias ni vacías**, y produce como salida una gramática equivalente a la de entrada, pero escrita en forma normal de Chomsky.

Considérese a modo de ejemplo la gramática definida por las siguientes producciones:

$$S \rightarrow aXbX \mid abX \mid aXb \mid ab$$

$$X \rightarrow aY \mid bY \mid a \mid b$$

$$Y \rightarrow aY \mid bY \mid a \mid b \mid c$$

El primer bucle del algoritmo 1 (líneas 1–8) introduce dos nuevos símbolos no terminales  $C_a$  y  $C_b$  y sus correspondientes producciones, de modo que la gramática se transforma en:

$$S \rightarrow C_a X C_b X \mid C_a C_b X \mid C_a X C_b \mid C_a C_b$$

$$X \rightarrow C_a Y \mid C_b Y \mid a \mid b$$

$$Y \rightarrow C_a Y \mid C_b Y \mid a \mid b \mid c$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

Finalmente el segundo bucle del algoritmo (líneas 9–16) transforma la gramática en:

$$S \rightarrow C_a D_1 \mid C_a E_1 \mid C_a F_1 \mid C_a C_b$$

$$X \rightarrow C_a Y \mid C_b Y \mid a \mid b$$

$$Y \rightarrow C_a Y \mid C_b Y \mid a \mid b \mid c$$

$$C_a \rightarrow a$$

$$C_b \rightarrow b$$

$$D_1 \rightarrow X D_2$$

$$D_2 \rightarrow C_b X$$

$$E_1 \rightarrow C_b X$$

$$F_1 \rightarrow X C_b$$

que ya está representada en forma normal de Chomsky. Estas referencias [5] [6] presentan ejemplos adicionales de conversión a CNF.

## 4. Ejercicio práctico

Desarrollar un programa `Grammar2CNF.cc` que lea un fichero `.gra` en el que figura la especificación de una gramática independiente del contexto y genere otro fichero `.gra` en el que se especifique una gramática equivalente ( $L(G_{in}) = L(G_{out})$ ) a la de entrada, pero en este caso, escrita en forma normal de Chomsky. Antes de realizar la conversión se deberá comprobar al menos que la gramática de entrada  $G_{in}$  no contiene producciones unitarias ni vacías. Por simplicidad, se podrá omitir la comprobación de símbolos y producciones inútiles. La conversión de la gramática de entrada  $G_{in}$  a forma normal de Chomsky se realizará mediante la aplicación del Algoritmo 1.

El comportamiento del programa al ejecutarse en línea de comandos debiera ser:

```
$ ./Grammar2CNF
```

```
Modo de empleo: ./Grammar2CNF input.gra output.gra
```

```
Pruebe 'Grammar2CNF --help' para más información.
```

Donde `input.gra` y `output.gra` son los ficheros que especifican las gramáticas de entrada y salida respectivamente. La opción `--help` en línea de comandos ha de producir que se imprima en pantalla un breve texto explicativo del funcionamiento del programa.

Los ficheros de especificación de gramáticas son ficheros de texto plano con extensión `.gra` que contienen los elementos definitorios de la gramática  $G \equiv (\Sigma, V, S, P)$  en este orden: símbolos terminales, símbolos no terminales, símbolo de arranque y producciones. El formato de cada uno de estos elementos en el fichero es el siguiente:

1. **Símbolos terminales (alfabeto)**: una línea que contiene  $N$ , el número de símbolos en el alfabeto seguida de  $N$  líneas, cada una de las cuales contiene un símbolo del alfabeto. Cada símbolo del alfabeto debe ser un único carácter imprimible.
2. **Símbolos no terminales**: una línea que contiene  $V$ , el número de símbolos no terminales, seguida de  $V$  líneas, cada una de las cuales contiene una cadena alfanumérica sin espacios. Se asumirá que el primero de los símbolos no terminales de esta lista será el **símbolo de arranque**.
3. **Producciones**: una línea que contiene  $P$ , el número de producciones de la gramática, seguida por  $P$  líneas cada una de las cuales contiene una producción en el formato:

A  $\alpha$

siendo  $\alpha \in (\Sigma \cup V)^*$ , es decir una secuencia de símbolos terminales y no terminales. La cadena vacía,  $\epsilon$  se representa mediante el carácter `&`.

Teniendo en cuenta la gramática utilizada para el ejemplo de conversión a forma normal de Chomsky, la especificación en un formato `.gra` sería la siguiente:

```
3
a
b
c
3
S
X
Y
13
S aXbX
S abX
S aXb
S ab
```

X aY  
X bY  
X a  
X b  
Y aY  
Y bY  
Y a  
Y b  
Y c

## 5. Criterios de evaluación

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que el profesorado tendrá en cuenta a la hora de evaluar el trabajo que el alumnado presentará en la sesión de evaluación de la práctica:

- Se valorará que el alumnado haya realizado, con anterioridad a la sesión de prácticas, y de forma efectiva, todas las tareas propuestas en este guión. Esto implicará que el programa compile y ejecute correctamente.
- También se valorará que, con anterioridad a la sesión de prácticas, el alumnado haya revisado los documentos que se enlazan desde este guión.
- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- El programa diseñado ha de ser fiel al paradigma de programación orientada a objetos. Se valorará que el alumnado haya identificado clases y objetos que permitan modelar adecuadamente el escenario de trabajo que se plantea.
- Se valorará que el programa se haya escrito de modo que las diferentes funcionalidades que se precisen hayan sido encapsuladas en métodos concretos cuya extensión textual se mantuviera acotada.
- El programa ha de ceñirse al formato de escritura de programas adoptado en las prácticas de la asignatura. Esto implica, entre otros, que el código desarrollado ha de estar escrito de acuerdo al estándar de la guía de estilo de Google para C++ [15].
- El programa desarrollado deberá compilarse utilizando la herramienta make [13, 14] y un fichero Makefile, separando los ficheros de declaración (\*.h) y definición (\*.cc) de clases.
- En la sesión de evaluación de este trabajo, todos los ficheros con código fuente se han de alojar en un único directorio junto con el fichero Makefile de compilación.
- Se requiere que todos los atributos de las clases definidas en el proyecto tengan un comentario descriptivo de la finalidad del atributo en cuestión. Además, se requiere que los comentarios del código fuente sigan el formato especificado por Doxygen [10].

Utilice asimismo esta [11] referencia para mejorar la calidad de la documentación de su código fuente.

- Finalmente, se analizará la capacidad del programador(a) de introducir cambios en el programa desarrollado.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

## Referencias

- [1] Transparencias del Tema 3 de la asignatura: Lenguajes y Gramáticas Independientes del Contexto, <https://campusingenieriaytecnologia2324.u11.es/mod/resource/view.php?id=4929>
- [2] Context Free Grammar, [https://en.wikipedia.org/wiki/Context-free\\_grammar](https://en.wikipedia.org/wiki/Context-free_grammar)
- [3] Chomsky Normal Form [https://en.wikipedia.org/wiki/Chomsky\\_normal\\_form](https://en.wikipedia.org/wiki/Chomsky_normal_form)
- [4] CYK algorithm [https://en.wikipedia.org/wiki/CYK\\_algorithm](https://en.wikipedia.org/wiki/CYK_algorithm)
- [5] Conversion of CFG to Chomsky Normal Form <https://tinyurl.com/cnf-example>
- [6] Chomsky Normal Form <https://courses.cs.washington.edu/courses/cse322/08au/lec14.pdf>
- [7] Noam Chomsky [https://en.wikipedia.org/wiki/Noam\\_Chomsky](https://en.wikipedia.org/wiki/Noam_Chomsky)
- [8] Chomsky hierarchy [https://en.wikipedia.org/wiki/Chomsky\\_hierarchy](https://en.wikipedia.org/wiki/Chomsky_hierarchy)
- [9] JavaScript 1.4 Grammar <https://www-archive.mozilla.org/js/language/grammar14.html>
- [10] Doxygen <http://www.doxygen.nl/index.html>
- [11] Diez consejos para mejorar tus comentarios de código fuente <https://www.genbeta.com/desarrollo/diez-consejos-para-mejorar-tus-comentarios-de-codigo-fuente>
- [12] Cómo identificar clases y objetos <http://www.comscigate.com/uml/DeitelUML/Deitel01/Deitel02/ch03.htm>
- [13] Makefile Tutorial: <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor>



[14] C++ Makefile Tutorial: <https://makefiletutorial.com>

[15] Google C++ Style Guide, <https://google.github.io/styleguide/cppguide.html>