Universidad de La Laguna

ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 8: Gramáticas Regulares y Autómatas Finitos

8.1. Objetivos

- Consolidar los conocimientos adquiridos sobre Gramáticas.
- Consolidar los conocimientos adquiridos sobre Autómatas Finitos.
- Estudiar y practicar el algoritmo de construcción de un NFA a partir de una Gramática Regular.
- Implementar en C++ una clase para representar Gramáticas.
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

8.2. Introducción

Desde la antiguedad, 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 metasímbolos 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* [1] 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. Las características de la sintaxis de los lenguajes naturales no se pueden modelar mediante CFGs.

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 [2]. 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* [3]. 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 [4], a modo de ejemplo, se utiliza una gramática independiente del contexto para especificar el lenguaje de descripición de gráficos DOT.

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 $G \equiv (V, \Sigma, S, P)$ cada uno de cuyos componentes se explican a continuación:

- V: Conjunto de símbolos no terminales ($V \neq \emptyset$)
- Σ: Conjunto de símbolos terminales (o alfabeto de la gramática), $\Sigma \cap V = \emptyset$
- S: Símbolo de arranque (***S**tart*) o axioma de la gramática ($S \in V$)
- P: Conjunto de reglas de producción:
  $P = \{A \to \alpha \mid A \in V,\ \alpha \in (V \cup \Sigma)^*\}$
  $P \subset V \times (V \cup \Sigma)^*$, ($P \neq \emptyset$)

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 aquél cuyas producciones aparecen en primer lugar en la lista de producciones.

Las siguientes producciones:

$S \to U \mid W$
$U \to TaU \mid TaT$
$W \to TbW \mid TbT$
$T \to aTbT \mid bTaT \mid \epsilon$

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. En el ejemplo:

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

Una Gramática $G \equiv (\Sigma, V, P, S)$ es lineal por la derecha si todas sus producciones tienen la forma:

$A \to uB \mid v$

Alternativamente, si todas las producciones son de la forma:

$A \to Bu \mid v$

La gramática es lineal por la izquierda. En estas expresiones $A, B \in V$ son símbolos no terminales y $u, v \in \Sigma^*$ son secuencias de símbolos terminales.

Una gramática es regular si es lineal por la izquierda o lineal por la derecha. Nótese que la restricción consiste en que las partes derechas de las reglas de producción han de contener un único símbolo no terminal y éste debe ser el primero (lineal por la izquierda) o el último símbolo (lineal por la derecha) de la regla.

Se presentan a continuación algunos ejemplos de gramáticas regulares:

- $G_1 \equiv (\Sigma, V, P, S)$ definida por: $\Sigma = \{a, b\}$ $V = \{S, A\}$ y las producciones:

  $S \to bA$
  $A \to aaA \mid b \mid \epsilon$

  Es una gramática regular (lineal por la derecha) que genera el lenguaje $L(G_1) = L(\texttt{b(aa)*b?})$: cadenas que comienzan con b, seguidas de un número par de aes y que pueden acabar también en b.

- $G_2 \equiv (\Sigma, V, P, S)$ definida por: $\Sigma = \{0, 1\}$ $V = \{S, A\}$ y las producciones:

  $S \to 0A$
  $A \to 10A \mid \epsilon$

  Es una gramática regular (lineal por la derecha) que genera el lenguaje $L(G_2) = L(\texttt{0(10)*})$.

- Una gramática regular, lineal por la izquierda que genera el mismo lenguaje es:
  $G_3 \equiv (\Sigma, V, P, S)$ definida por: $\Sigma = \{0, 1\}$ $V = \{S\}$ y las producciones:

  $S \to S10 \mid 0$

  L(G_3)= L(G_2)

Dada una Gramática Regular, $G \equiv (V, \Sigma, S, P)$ siempre es posible hallar un NFA $M \equiv (Q, \Sigma', \delta, q_0, F)$ tal que $L(M) = L(G)$. El algoritmo de construcción de ese NFA procede como se describe a continuación.

Se define inicialmente el NFA como:

- $Q = V \cup \{f\}$ donde f es el único estado de aceptación del NFA ($f \notin V$)
- $F = \{f\}$
- $\Sigma' = \Sigma$
- $q_0 = S$

Y se añaden transiciones y estados adicionales a Q siguiendo estas reglas:

- Si $(A \to a_1 a_2 \dots a_n B) \in P \Rightarrow Q = Q \cup \{q_1, q_2, \dots, q_{n-1}\}$
  y se añaden también las transiciones:
  $\delta(A, a_1 a_2 ... a_n) = \delta(q_1, a_2 ... a_n) = \dots = \delta(q_{n-1}, a_n) = B$
- Si $(A \to c_1 c_2 \dots c_m) \in P \Rightarrow Q = Q \cup \{p_1, p_2, \dots, p_{m-1}\}$
  y se añaden las transiciones
  $\delta(A, c_1 c_2 \dots c_m) = \delta(p_1, c_2 \dots c_m) = \dots = \delta(p_{m-1}, c_m) = f$

Para el caso de la Gramática Regular G_2:

$S \to 0A$
$A \to 10A \mid \epsilon$

el NFA resultante es el que se muestra en la Figura 8.1.

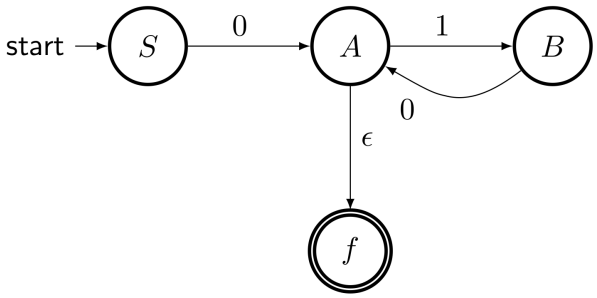

Figura 8.1: NFA que reconoce el lenguaje representado por la expresión `0(10)*`

Tal como se ha indicado anteriormente la presencia de una producción que en la parte derecha tiene *n* símbolos terminales, $(A \to a_1 a_2 \dots a_n B)$ provoca la introducción de *n-1* estados adicionales. En el caso del ejemplo anterior, la presencia de la producción $A \to 10A$ provoca que se introduzca un nuevo estado, B además de los inciales $\{S, A, f\}$.

En este documento [5] se puede ver un ejemplo adicional ilustrado con JFlap.

8.3. Ejercicio práctico

Desarrollar un programa `G2NFA.cpp` que lea un fichero de texto en el que figura la especificación de una Gramática Regular lineal por la derecha, G y genere un fichero `.nfa` con la especificación de un NFA M, tal que L(G) = L(M).

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

```
$ ./G2NFA
Modo de empleo: ./G2NFA input.gra output.nfa
Pruebe 'G2NFA --help' para más información.
```

Donde `input.gra` es el fichero conteniendo la descripción de la gramática y `output.nfa` es el fichero de especificación del NFA. 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` El Listado 8.1 muestra el fichero de especificación de la gramática G_2. Estos ficheros 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 caracter imprimible.

2. Conjunto de 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.

3. Símbolo de arranque: una única línea que contiene el símbolo de arranque, S, de la gramática. Ha de ser uno de los símbolos no terminales relacionados anteriormente.

4. 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 -> ` α

   siendo $\alpha \in (\Sigma \cup V)^*$, es decir una secuencia de símbolos terminales y no terminales. La cadena vacía, ϵ se representa mediante el carácter ~ (código ASCII 126).

Todas las líneas de un fichero `.gra` que comiencen con los caracteres `//` corresponden a comentarios, y deben ser ignorados por el programa a la hora de procesar el fichero.

```
1  // Universidad de La Laguna
2  // Grado en Ingenieria Informatica
3  // Computabilidad y Algoritmia
4  // Fichero de representacion de la Gramatica:
5  //
6  //       S -> 0A
7  //       A -> 10A | epsilon
8  //
9  // Lenguaje generado: 0(10)*
10 //
11 ////////////////////////////////////
12 2
13 0
14 1
15 2
16 S
17 A
18 S
19 3
20 S -> 0A
21 A -> 10A
22 A -> ~
```

Listado 8.1: Fichero de especificación de la gramática G_2

Los ficheros de especificación de NFAs definen en este orden el alfabeto, los estados, el estado inicial, los estados finales y las transiciones. El formato de cada uno de estos elementos en el fichero es el siguiente:

1. Alfabeto: una línea que contiene N, el número de símbolos en el alfabeto seguida de N+1 líneas. Una ϵ-transición se representará mediante el carácter ~ (código ASCII 126). De las N+1 líneas, la primera de ellas contiene el símbolo correspondiente a ϵ (caracter ~) y las N siguientes se corresponden con cada uno de los símbolos del alfabeto. Cada símbolo del alfabeto debe ser una cadena de caracteres imprimibles, sin incluir espacios en blanco.

2. Estados: una línea que contiene M, el número de estados, seguida de M líneas, cada una de las cuales contiene el identificador (etiqueta) de un estado. Cada identificador de estado debe ser una cadena alfanumérica sin espacios.

3. Estado inicial: una línea que contiene el identificador del estado inicial. El estado inicial ha de ser uno de los estados relacionados anteriormente.

4. Estados de aceptación: una línea que contiene F, el número de estados finales, seguido por F líneas, cada una de las cuales contiene el identificador de un estado final. Cada estado final ha de ser uno de los relacionados anteriormente.

5. Transiciones: una línea que contiene T, el número de transiciones en la función δ, seguida de T líneas, cada una de las cuales contiene tres cadenas *qi*, *sim*, *qf*

separadas por espacios. Cada una de estas líneas indica una transición del estado *qi* al *qf* con símbolo *sim*, es decir, $\delta(qi, sim) = qf$. *qi* y *qf* han de estar listados en la lista de estados y *sim* debe estar en la lista de símbolos del alfabeto o ser el carácter ~ para el caso de una ϵ-transición.

El programa `G2NFA.cpp` ha de contener al menos las clases `Grammar` y `Nfa` para representar gramáticas y NFAs respectivamente. La clase `Grammar` ha de contemplar un constructor que tome como parámetro el nombre del fichero de especificación de la gramática, así como un método *convertToNFA()* que devuelva como salida un objeto de la clase `Nfa`. La clase `Nfa` ha de definir un método al que, pasándole un nombre de fichero imprima en ese fichero la especificación del NFA correspondiente.

8.4. Rúbrica de evaluación del ejercicio

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:

- Capacidad del programador(a) de introducir cambios en el programa desarrollado.
- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- El programa diseñado ha de seguir el paradigma OOP.
- Se valorará la coherencia e idoneidad del diseño realizado en las clases diseñadas para representar gramáticas y autómatas finitos.
- Modularidad: el programa ha de escribirse de modo que las diferentes funcionalidades que se precisen sean encapsuladas en métodos cuya extensión textual se mantenga acotada.
- El programa ha de ceñirse al formato de escritura de programas adoptado en las prácticas de la asignatura.
- En la sesión de evaluación de este trabajo, todos los ficheros con códido fuente se han de alojar en un único directorio junto con el fichero Makefile de compilación.
- Todos los atributos de las clases definidas en el proyecto han de tener un comentario descriptivo de la finalidad del atributo en cuestión.
- Se valorará que los comentarios del código fuente sigan el formato de los comentarios de Doxygen.

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.

Bibliografía

[1] Context Free Grammar https://en.wikipedia.org/wiki/Context-free_grammar

[2] Noam Chomsky https://en.wikipedia.org/wiki/Noam_Chomsky

[3] Chomsky hierarchy https://en.wikipedia.org/wiki/Chomsky_hierarchy

[4] The DOT Language. https://www.graphviz.org/doc/info/lang.html

[5] Converting Regular Grammar to DFA http://www.jflap.org/modules/ConvertedFiles/Regular%20Grammar%20to%20DFA%20Conversion%20Module.pdf