Lenguajes y gramáticas independientes del contexto

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

03-2021

Contenido I

Definición de las gramáticas independientes del contexto

Existen cuatro componentes importante de una descripción gramatical de un lenguaje:

1. Un conjunto finito de símbolos que forma las cadenas del lenguaje que se está definiendo. Denominamos a este conjunto *alfabeto terminal* o *alfabeto de símbolos terminales*.
2. Un conjunto finito de variables, denominado también en ocasiones *símbolos no terminales* o *categorías sintácticas*. Cada variable representa un lenguaje; es decir, un conjunto de cadenas.
3. Una de estas variables representa el lenguaje que se está definiendo; denominada *símbolo inicial*.

4. Un conjunto finito de *producciones* o *reglas* que representan la definición recursiva de un lenguaje. Cada producción consta de:
   - 4.1 Una variable a la que define (parcialmente) la producción. Esta variable, a menudo, se denomina *cabeza* de la producción.
   - 4.2 El símbolo de producción $\rightarrow$.
   - 4.3 Una cadena formada por cero o más símbolos terminales y variables (no terminales). Esta cadena, denominada *cuerpo* de la producción, representa una manera de formar cadenas pertenecientes al lenguaje de la variable de la cabeza.

Los cuatros componentes que acabamos de describir definen una gramática independiente del contexto, GIC o simplemente una *gramática* (*context-free grammar, CFG*). Representaremos una GIC G mediante sus cuatro componentes, es decir, $G = (V, T, P, S)$, donde V es el conjunto de variables (terminales), T son los símbolos terminales, P es el conjunto de producciones y S es el símbolo inicial.

Un ejemplo...

Consideremos el lenguaje de los palíndromos. Un *palíndromo* es una cadena que se lee igual de izquierda a derecha que de derecha a izquierda. Dicho de otra manera, la cadena w es un palíndromo si y sólo si $w = w^R$. Para hacer las cosas más sencillas, consideraremos únicamente los palíndromos descritos con el alfabeto $\{0, 1\}$.
Una definición recursiva y natural, sería:

- **Base**. ϵ, 0 y 1 son palíndromos.
- **Paso Inductivo**. Si w es un palíndromo, también lo son $0w0$ y $1w1$.

Ninguna cadena es un palíndromo de ceros y unos, a menos que

Ahora, un ejemplo más complejo...

Ahora, definiremos una gramática que representa la simplificación de las expresiones de un lenguaje de programación típico. En primer lugar, vamos a limitarnos a los operadores $+$ y $*$, que representan la suma y la multiplicación, respectivamente. Establecemos que los argumentos son identificadores definidos como letras *a* y *b* seguidas por cero o más letras *a* y *b* o dígitos 0 y 1.

1. $E \rightarrow I$
2. $E \rightarrow E+E$
3. $E \rightarrow E*E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Derivaciones utilizando una gramática

Empleamos las producciones de una GIC para si ciertas cadenas pertenecen al lenguaje una cierta variable. Para llevar a cabo esta inferencia hay dos métodos disponibles.

- El más convencional consiste en emplear las reglas para pasar del cuerpo a la cabeza. Es decir, tomamos cadenas que sabemos que pertenece al lenguaje de cada una de las variable del cuerpo, las concatenamos en el orden apropiado con cualquier símbolo terminal que aparezca en el cuerpo e inferimos que la cadena resultante pertenece al lenguaje de la variable de la cabeza. Este procedimiento lo denominaremos *inferencia recursiva*.

	Cadena inferida	Para el lenguaje de	Producción usada	Cadena(s) usada(s)
(i)	a	I	5	—
(ii)	b	I	6	—
(iii)	$b0$	I	9	(ii)
(iv)	$b00$	I	9	(iii)
(v)	a	E	1	(i)
(vi)	$b00$	E	1	(iv)
(vii)	$a+b00$	E	2	$(v), (vi)$
$(viii)$	$(a+b00)$	E	4	(vii)
(ix)	$a*(a+b00)$	E	3	$(v), (viii)$

- Existe otro método que permite definir el lenguaje de una gramática en el que se emplean las producciones desde la cabeza hasta el cuerpo. El símbolo inicial se expande utilizando una de sus producciones (es decir, mediante una producción cuya cabeza sea el símbolo inicial). A continuación, expandimos la cadena resultante reemplazando una de las variables por el cuerpo de una de sus producciones, y así sucesivamente, hasta obtener una cadena compuesta totalmente por terminales.

Notación para las derivaciones de las GIC

Existe una serie de convenios de uso común que nos ayudan a recordar la función de los símbolos utilizados al tratar con las GIC. Los convenios que emplearemos son los siguientes:

1. Las letras minúsculas del principio del alfabeto (a, b, c, etc. son símbolos terminales. También supondremos que los dígitos y otros caracteres como el signo más ($+$) o los paréntesis también son símbolos terminales.
2. Las letras mayúsculas del principio del alfabeto (A, B, C, etc.) son variables.
3. Las letras minúsculas del final del alfabeto, como w o z, son cadenas de símbolos terminales.
4. Las letras mayúsculas del final del alfabeto, como X o Y, son símbolos terminales o variables.
5. Las letras griegas minúsculas, como α o β, son cadenas formadas por símbolos terminales y/o variables.

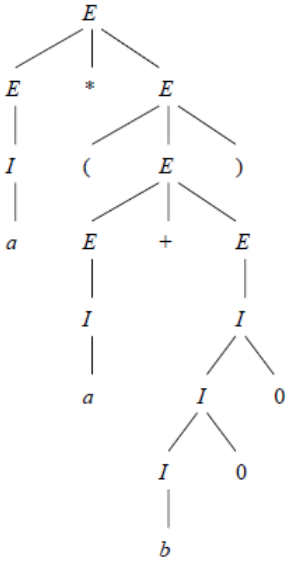
Derivaciones izquierda y derecha

- Con el fin de restringir el número de opciones disponibles en la derivación de una cadena, a menudo resulta útil requerir que en cada paso se reemplace la variable más a la izquierda por uno de los cuerpos de sus producciones. Tal derivación se conoce como *derivación más a la izquierda*.
- De forma similar, se puede hacer que en cada paso se reemplace la variable más a la derecha por uno de los cuerpos de sus producciones. En este caso, se trata de una *derivación más a la derecha*.

Construcción de los árboles de derivación

Sea $G = (V, T, P, S)$ una gramática. Los árboles de derivación para G son aquellos árboles que cumplan con las siguientes condiciones:

1. Cada nodo interior está etiquetado como una variable de V.
2. Cada hoja está etiquetada con una variable, un símbolo terminal o ϵ. Sin embargo, si la hoja está etiqueta con ϵ, entonces tiene que ser el único hijo de su padre.
3. Si un nodo interior está etiquetado como A y sus hijos están etiquetados como $X_1, X_2, ..., X_k$ respectivamente, comenzado por la izquierda, entonces $A \to X_1X_2...X_k$ es una producción de P. Observe que el único caso en que una de las X puede reemplazarse por ϵ es cuando es la etiqueta del único hijo y $A \to \epsilon$ es una producción de G.


E
E
*
E
I
(
E
)
a
E
+
E
I
I
a
I
0
I
0
b

Aplicaciones de las gramáticas independientes de contexto

Las gramáticas independientes del contexto originalmente fueron concebidas por N. Chomsky como una forma de describir los lenguajes naturales. Pero esta posibilidad no ha llegado a cumplirse. Vamos a describir dos posibles usos, uno antiguo y otro nuevo:

1. Las gramáticas se utilizan para describir lenguajes de programación. Lo más importante es que existe una forma mecánica de convertir la descripción del lenguaje como GIC es un analizador sintáctica, el componente del compilador que descubre la estructura del programa fuente y representa dicha estructura mediante un árbol de derivación. Esta aplicación constituye uno de los usos más tempranos de la GIC; de hecho, es una de las primera formas en las que las ideas teóricas de las Ciencias Computacionales pudieron llevarse a la práctica.

2. El desarrollo de XML (Extensible Markup Language), lenguage que facilita el comercio electrónico permitiendo el intercambio de información. Una parte fundamental del XML es la *definición de tipo de documento* (DTD, Document Type Definition), que principalmente es una gramática independiente del contexto que describe las etiquetas permitidas y las formas en que dichas etiquetas pueden anidarse.

Analizadores sintácticos

Muchos de los aspectos de un lenguaje de programación tienen una estructura que puede describirse mediante expresiones regulares. Por ejemplo, podemos representar identificadores mediante expresiones regulares. Sin embargo, también hay algunos aspectos importantes de los lenguajes de programación típicos que no pueden representarse sólo mediante expresiones regulares.

Los lenguajes típicos emplean paréntesis y/o corchetes de forma equilibrada y anidada. Es decir, hay que emparejar un paréntesis abierto por la izquierda con el paréntesis de cierra que aparece inmediatamente a su derecha, eliminar ambos y repetir el procesos. Si al final se eliminan todos los paréntesis, entonces la cadena estaba equilibrada y si no se pueden emparejar los paréntesis de esta manera, entonces es que estaba desequilibrada. Ejemplos de cadenas con paréntesis equilibrados son (()), ()(), (() ()), y ϵ, mientras que)(y (() no lo son.

$$B \to BB$$
$$B \to (B)$$
$$B \to \{B\}$$
$$B \to [B]$$
$$B \to \epsilon$$

Por supuesto, los lenguajes de programación constan de algo más que paréntesis, aunque estos son una parte fundamental de las expresiones artiméticas y condicionales.

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow number$$
$$F \rightarrow (E)$$

Muchos aspectos de un lenguaje de programación típico se comportan como los paréntesis equilibrados entre ellos los propios paréntesis que se utilizan en expresiones de todo tipo. Algunos ejemplos son los elementos que marcan el principio y el final de los bloques de código, como **begin** y **end** en Pascal, o las llaves {.. en C.

Existe un caso que se presente ocasionalmente en el que los "paréntesis"pueden ser equilibrados, pero también pueden existir paréntesis abiertos no equilibrados. Un ejemplo sería el tratamiento de **if** y **else** en C. Puede existir una cláusula **if** desequilibrada o equilibrada por la cláusula **else** correspondiente. Una gramática que genera las secuencias posibles de **if** y **else** (representadas por *i* y *e*, respectivamente) es:

$$S \rightarrow iCtS$$
$$S \rightarrow iCtSe$$
$$S \rightarrow SS$$
$$S \rightarrow \epsilon$$

El generador de analizadores YACC

La generación de un analizador (función que crea los árboles de derivación a partir de los programas fuente) ha sido institucionalizada por la aplicación YACC (y derivados). La entrada a YACC es una GIC, con una notación que sólo difiere en algunos detalles respecto de la que hemos empleado aquí. Con cada producción se asocia una *acción*, que es un fragmento de código C que se ejecuta cuando se crea un nodo del árbol de derivación, el cual (junto con sus hijos) corresponde a esta producción. Normalmente, la acción es el código para construir dicho nodo, aunque en algunas aplicaciones YACC el árbol no se construye realmente y la acción hace algo más, por ejemplo, generar un fragmento del código objeto.

```
Exp : Id            {...}
    | Exp '+' Exp   {...}
    | Exp '*' Exp   {...}
    | '(' Exp ')'   {...}
    ;
Id  : 'a'           {...}
    | 'b'           {...}
    | Id 'a'        {...}
    | Id 'b'        {...}
    | Id '0'        {...}
    | Id '1'        {...}
    ;
```

Lenguajes de marcado

- A continuación vamos a ocuparnos de una familia de "lenguajesçonocidos como lenguajes de marcado. Las “cadena” de estos lenguajes son documentos con determinadas marcas (denominadas *etiquetas*). Las etiquetas nos informan acerca de las semántica de las distintivas cadenas contenidas en el documento.
- El lenguaje de marcado con el que probablemente estés más familiarizado es el HTML (*HyperText Markup Language*). Este lenguaje tiene dos funciones pricinpales: crear vínculos entre documentos y describir el formato (aspecto) de un documento.

Las cosas que *odio*:

1. Pan mohoso.
2. La gente que conduce muy despacio en una autovía.

(a) El texto tal y como se visualiza.

```
<P>Las cosas que  <EM>odio</EM>:
<OL>
<LI>Pan mohoso.
<LI>La gente que conduce muy despacio
en una autovía.
</OL>
```

(b) El código fuente HTML.

Existe una serie de clases de cadenas asociadas con un documento HTML. No vamos a enumerarlas todas, sólo vamos a citar aquellas que son fundamentales para comprender textos como el mostrado en la imagen anterior. Para cada clase utilizaremos una variable con un nombre descriptivo:

1. *Texto* es cualquier cadena de caracteres que se puede interpretar de forma literal, es decir, que no contiene etiquetas. Por ejemplo, "pan mohoso".
2. *Car* es cualquier cadena que consta de un solo carácter que es válido en un texto HTML. Por ejemplo, los espacios en blanco.
3. *Doc* representa documentos, que son secuencias de "elementos". Definimos los elementos a continuación y dicha definición es mutuamente recursiva con la definición de un *Doc*.

4. *Elemento* es o una cadena de *Texto* o un par de etiquetas emparejadas y el documento que haya entre ellas, o una etiqueta no emparejada seguida de un documento.
5. *ListItem* es la etiqueta <LI> seguida de un documento, que es simplemente un elemento de una lista.
6. *Lista* es una secuencia de cero o más elementos de una lista.

1. $Car \rightarrow a \mid A \mid \cdots$
2. $Texto \rightarrow \varepsilon \mid Car\ Texto$
3. $Doc \rightarrow \varepsilon \mid Elemento\ Doc$
4. $Elemento \rightarrow Texto \mid$
   <EM> Doc </EM> $\mid$
   <P> $Doc \mid$
   <OL> $Lista$ </OL> $\mid \cdots$
5. $ListItem \rightarrow$ <LI> Doc
6. $Lista \rightarrow \varepsilon \mid ListItem\ Lista$

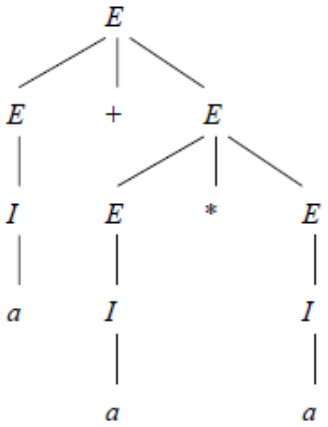
Ambigüedad en gramáticas y lenguajes

Como hemos visto, las aplicaciones de las GIC a menudo confían en la gramática para proporcionar la estructura de los archivos. Por lo tanto, la suposición tácita es que una gramática determina de manera inequívoca una estructura para cada cadena del lenguaje. Sin embargo, veremos que no todas las gramáticas proporcionan estructuras únicas.

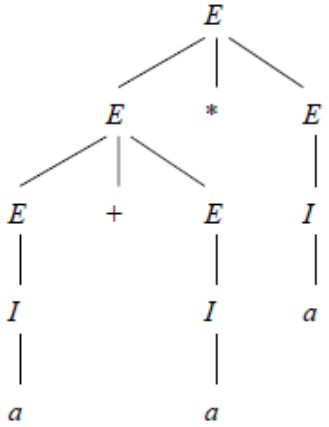
Desarrolla el árbol de derivación de la cadena:
$a + a * a$

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$



(a)



(b)

En un mundo ideal, podríamos proporcionar un algoritmo para eliminar la ambigüedad de las GIC. Pero, no existe un algoritmo que nos diga si una GIC es ambigua.

Afortunadamente, la situación en la práctica no es tan sombría. Para los tipos de construcciones que aparecen en los lenguajes de programación comunes, existen técnicas bien conocidas que permiten eliminar la ambigüedad.

En primer lugar, observamos que existen dos causas de ambigüedad en la gramática:

1. La precedencia de los operadores no se respeta.
2. Una secuencia de operadores idénticos puede agruparse empezando por la izquierda o por la derecha.