Autómatas y Lenguajes Formales 2017-1
Facultad de Ciencias UNAM
Nota de Clase 12

Favio E. Miranda Perea A. Liliana Reyes Cabello Lourdes González Huesca

19 de octubre de 2016

Como hemos discutido, los lenguajes libres de contexto engloban lenguajes más complejos o expresivos que los lenguajes regulares, sin embargo esta característica hace que algunos problemas de decisión sean más difíciles de resolver o incluso que no sean decidibles.
En esta nota abordaremos algunos problemas de decisión sobre lenguajes libres de contexto.

1. Ambigüedad

Está probado que no puede existir un algoritmo que determine con certeza si una gramática es ambigua o no, y que en tal caso elimine dicha ambigüedad produciendo una gramática no ambigua equivalente a la original.

Es decir, el problema de ambigüedad es indecidible. Lo más que se puede saber es que hay ciertas condiciones que determinan ambigüedad pero en caso de no cumplirse estas nada puede decirse a cerca de la gramática en cuestión. En algunos casos, dada una gramática ambigua, se puede encontrar otra gramática equivalente no ambigua, por ejemplo agregando precedencia de símbolos u operadores y asociatividad. Sin embargo existen lenguajes cuya ambigüedad es inevitable.

Ejemplo: Un lenguaje compacto para expresiones aritméticas es generado por la siguiente gramática ambigua

$$S \to S + S \mid S * S \mid a$$

donde a es un terminal que representa a los identificadores y constantes.

Podemos demostrar la ambigüedad de la gramática con la cadena $a + a * a$ que tiene dos árboles de derivación: si se empieza con la producción $S \to S + S$ o con la producción $S \to S * S$.

Una gramática no ambigua equivalente se obtiene modelando la precedencia de operadores como sigue:

$$\begin{array}{lcll} S & \to & S + T \mid T & \\ T & \to & T * F \mid F & \text{la multiplicación tiene mayor precedencia} \\ F & \to & (S) \mid a & \text{agregar paréntesis limita la precedencia} \end{array}$$

1.1. Gramáticas libres de contexto en lenguajes de programación

El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica:

- La semántica se encarga de definir el significado de las expresiones, enunciados y unidades de programa.
- La pragmática define la implementación del lenguaje basada en la metodología y estrategias de programación deseadas.
- La sintaxis se encarga de definir la forma de las expresiones y enunciados de un lenguaje y se sirve fundamentalmente de los conceptos y herramientas de nuestro curso.

Antes del proceso de evaluación, un compilador e intérprete necesita realizar los procesos de análisis léxico y sintáctico, los describimos a continuación a grandes rasgos:

- El análisis léxico se encarga de transformar el programa fuente en una lista de unidades sintácticas de bajo nivel llamadas *lexemas*, los cuales se clasifican en distintas categorías llamadas **tokens**, como pueden ser identificadores, constantes, separadores, etc.
  El análisis léxico se sirve fundamentalmente de expresiones regulares para su definición y reconocimiento.
- El análisis sintáctico o *parsing* se encarga de transformar la lista de lexemas en un programa objeto, el cual es una expresión válida de la llamada sintaxis abstracta del lenguaje.
  El programa objeto es esencialmente un árbol de derivación dictado por una gramática libre de contexto que define al lenguaje de programación.

Por lo tanto el análisis sintáctico es esencialmente una forma del problema de la pertenencia en gramáticas libres de contexto.

Forma de Backus-Naur Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o **BNF**. Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60. Este sistema notacional para definir lenguajes libres de contexto sigue las siguientes convenciones:

- El símbolo de reescritura $\rightarrow$ se reemplaza con ::=.
- El símbolo | significa *ó* y se usa para abreviar las producciones de una misma variable.
- Las variables se escriben entre paréntesis triangulares y por lo general utilizan nombres largos que ayuden a la descripción de las categorías del lenguaje.

Ejemplo: Lenguaje de paréntesis balanceados

$$\begin{aligned} < parent\_balanc > \quad ::= \quad & \varepsilon \mid \\ & (< parent\_balanc >) \mid \\ & < parent\_balanc >< parent\_balanc > \end{aligned}$$

Por ejemplo la cadena ()()() tiene dos árboles de derivación.
Una gramática no ambigua equivalente es: $S \rightarrow \varepsilon \mid (S)S$ y en forma **BNF**:

$$< parent\_balanc > ::= \varepsilon \mid (< parent\_balanc >) < parent\_balanc >$$

Ejemplo: El lenguaje de las expresiones aritméticas

$$\begin{aligned} < expr > &::= < expr >< op >< expr > | (< expr >) | < id > \\ < op > &:= + | - | * | / \\ < id > &:= a | b | c \end{aligned}$$

Extendido para generar bloques de asignación de la forma:

`begin` a `:=` b/c ; b `:=` a*(b+c) `end`

$$\begin{aligned} < programa > &::= \texttt{begin} < sec\_enunc > \texttt{end} \\ < sec\_enunc > &::= < enunc > | < enunc > ; < sec\_enunc > \\ < enunc > &::= < id > := < expr > \\ < expr > &::= < expr >< op >< expr > | (< expr >) | < id > \\ < op > &:= + | - | * | / \\ < id > &:= a | b | c \end{aligned}$$

Ejemplo: El lenguaje de las expresiones condicionales:

$$\begin{aligned} < enunc > &::= < condicional > | \ldots \\ < condicional > &:= \texttt{if} < expr > \texttt{then} < enunc > | \\ &\quad\ \texttt{if} < expr > \texttt{then} < enunc > \texttt{else} < enunc > \end{aligned}$$

La ambigüedad de este lenguaje se presenta con problema llamado del `if` colgante:

```
if false then if false then 0 else 1
```

Este código tiene dos significados:

```
if false then (if false then 0) else 1
if false then (if false then 0 else 1)
```

Una gramática equivalente no ambigua es:

$$\begin{aligned} S &\rightarrow C \mid O \\ C &\rightarrow C1 \mid C2 \\ C1 &\rightarrow \texttt{if}\ E\ \texttt{then}\ C1\ \texttt{else}\ C1 \\ C2 &\rightarrow \texttt{if}\ E\ \texttt{then}\ S \mid \texttt{if}\ E\ \texttt{then}\ C1\ \texttt{else}\ C2 \end{aligned}$$

La idea detrás de esta gramática es separar los posibles condicionales anidados:

- $C1$ genera condicionales dobles (if-then-else) balanceados
- $C2$ representa condicionales simples (if-then) y condicionales dobles pero de forma que un `if` $-$ `then` sólo figura colgando al final es decir en el `else`.

2. Problema de pertenencia

Dado un lenguaje libre de contexto L y una cadena $w \in \Sigma^\star$, ¿cómo decidir si $w \in L$?

Para resolver esta pregunta, por un lado se puede generar un autómata no-determinista que reconozca el lenguaje, en este caso las derivaciones que pueden generarse son exponenciales.

Por otro lado si se utilizara una gramática libre de contexto, se puede optar por normalizarla y obtener una equivalente en forma normal de Chomsky o de Greibach. Así se pueden generar todas la derivaciones de longitud $2|w|-1$ y obtener una derivación que produzca exactamente a la cadena w.

Pero en general, si se quiere resolver el problema de decisión anterior de manera eficiente las opciones anteriores no son las mejores.

En la década de 1960 se desarrollaron varias aportaciones a los lenguajes de programación y a los compiladores de los mismos, entre ellas se encuentra un algoritmo diseñado por J. Cocke, T. Kasami y D.H. Younger para realizar el análisis sintáctico de una cadena dado un lenguaje.

Este algoritmo llamado **CYK** pertenece a la programación dinámica cuyo lema es "divide y vencerás": utiliza una gramática libre de contexto normalizada y realiza un análisis de abajo hacia arriba (*bottom-up*) para revisar la cadena y las subcadenas de ella.

Este algoritmo es sobresaliente dada la eficiencia bajo ciertas características: en el peor de los casos tiene un tiempo de $\Theta(n^3 \cdot |V|)$ donde n es la longitud de la cadena a procesar y $|V|$ es el número de variables en la gramática.

Formalemente, el algoritmo recibe como entrada una gramática en forma normal de Chomsky y una cadena $w = a_1 a_2 \cdots a_n$. El procedimiento analiza la cadena por partes, es decir, las subcadenas y reglas que pueden generar esas subcadenas.
Para este análisis se forman conjuntos $X_{i,j}$, que continen símbolos no terminales correspondientes a la parte izquierda de una producción que genera la subcadena:

$$A \in X_{i,j} \text{ si } A \to^+ a_i a_{i+1} \cdots a_{i+j-1}$$

Si el conjunto $X_{1,n}$ correspondiente a la longitud de la cadena contiene al símbolo inicial de la gramática entonces se puede decir que la cadena w pertenece al lenguaje generado por la gramática.

El pseudo-algoritmo para generar los conjuntos $X_{i,j}$ es el siguiente:

1. Formar una cuadrícula o una matriz de $n \times n$:
   - cada espacio de la matriz será el conjunto $X_{i,j}$ correspondiente a sus coordenadas, los renglones (j) están enumerados de abajo hacia arriba y las columnas (i)de izquierda a derecha ambos desde 1 hasta n.
   - en la parte inferior de la cuadrícula colocar por cada columna un símbolo de la cadena
   - el renglón 1 (el que está más abajo) es llenado con los símbolos no terminales que puedan generar el símbolo que se encuentra debajo de cada celda.
2. Para cada una de las celdas de los siguientes renglones ($X_{i,j}$ con $i, j > 1$) agregar los símbolos A de las producciones $A \to BC$ donde $B \in X_{i,k}$ y $C \in X_{i+k,j-k}$ y $k < j - 1$.

Ejemplo: Dada la cadena $w = bbab$ y la gramática G en **FNC**, decir si $w \in L(G)$.

$$\begin{array}{rcl} S & \rightarrow & BA \mid AC \\ A & \rightarrow & CC \mid b \\ B & \rightarrow & AB \mid a \\ C & \rightarrow & BA \mid a \end{array}$$

1. Inicializamos la tabla con la longitud de la cadena y el renglón $j = 1$ con las variables que generan los símbolos

$j = 4$				
$j = 3$				
$j = 2$				
$j = 1$	A	A	B, C	A
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	b

2. Para $j = 2$ tenemos las producciones para subcadenas de longitud dos

$j = 4$				
$j = 3$				
$j = 2$	-	B, S	S, C	
$j = 1$	A	A	B, C	A
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	b

3. Para $j = 3$ tenemos las producciones para subacadenas de longitud tres, en el caso particular del conjunto $X_{2,3}$ se utilizan los conjuntos por debajo de él que generan las subcadenas:

$j = 4$				
$j = 3$	B	**S, C**		
$j = 2$	-	**B, S**	**S, C**	
$j = 1$	A	**A**	B, C	**A**
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	b

4. Y finalmente la tabla completa

$j = 4$	**S, C**			
$j = 3$	**B**	**S, C**		
$j = 2$	**-**	B, S	**S, C**	
$j = 1$	**A**	A	B, C	**A**
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	b

En el conjunto $X_{1,4}$, que representa la cadena completa, está contenido el símbolo inicial de la gramática y por lo tanto w pertenece a $L(G)$. Para poder obtener una derivación de w sólo es necesario tomar los símbolos correspondientes que generan las subcadenas:

$$S \to AC \to aC \to bBA \to bABA \to bbBA \to bbaA \to bbab$$

Ahora analicemos la cadena $w = bbaa$, tenemos las siguientes tablas:

1. Inicializamos la tabla con $j = 1$

$j = 4$				
$j = 3$				
$j = 2$				
$j = 1$	A	A	B, C	B, C
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	a

2. Para $j = 2$ tenemos las producciones para subcadenas de longitud dos

$j = 4$				
$j = 3$				
$j = 2$	-	B, S	A	
$j = 1$	A	A	B, C	B, C
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	a

3. Para $j = 3$ tenemos las producciones para subacadenas de longitud tres

$j = 4$				
$j = 3$	B	-		
$j = 2$	-	B, S	A	
$j = 1$	A	A	B, C	B, C
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	a

4. Y finalmente la tabla completa

$j = 4$	-			
$j = 3$	B	-		
$j = 2$	-	B, S	A	
$j = 1$	A	A	B, C	B,C
	$i = 1$	$i = 2$	$i = 3$	$i = 4$
	b	b	a	a

Como podemos ver el conjunto $X_{1,4}$ es vacío, por lo tanto la cadena no es aceptada por la gramática.

3. Lenguajes que no son libres de contexto

Análogamente al caso de lenguajes regulares, las dos formas principales para demostrar que un lenguaje no es libre de contexto son el uso de las propiedades de cerradura y el lema del bombeo, el cual enunciamos más adelante.
Recordemos las propiedades de cerradura de lenguajes libres de contexto:

- Unión: si L_1, L_2 son lenguajes libres del contexto entonces $L_1 \cup L_2$ es libre del contexto.
- Concatenación: si L_1, L_2 son lenguajes libres del contexto entonces $L_1 L_2$ es libre del contexto.
- Estrella de Kleene: si L_1 es un lenguaje libres del contexto entonces $L_1^\star$ es libre del contexto.

Estas propiedades permiten construir lenguajes libres de contexto y también proveen un mecanismo para demostrar si alguno lo es. El siguiente lema es útil al **refutar** que un lenguaje sea libre de contexto.

Lema 1 (Lema del Bombeo para Lenguajes libres de contexto) *Si L es un lenguaje libre de contexto entonces existe un número $n \in \mathbb{N}$ llamado constante de bombeo para L, tal que para cualquier cadena $z \in L$ con $|z| \geq n$ existen cadenas u, v, w, x, y tales que $z = uvwxy$ y además*

- $|vwx| \leq n$
- $|vx| \geq 1$, *es decir* $v \neq \varepsilon$ *ó* $x \neq \varepsilon$ *pero no ambas*
- *para todo* $i \geq 0$, $uv^i wx^i y$ *debe pertenecer al lenguaje*

Demostración. La demostración se basa en un análisis de árboles de derivación de gramáticas en forma normal de Chomsky.

Veamos algunos ejemplos del uso del lema del bombeo:

Ejemplo: Sea $L = \{a^i b^i c^i \mid i \in \mathbb{N}\}$. Demostrar que L **no** es libre de contexto.
Supóngase que L es libre del contexto y sea n una constante de bombeo. Entonces sea $z = a^n b^n c^n$ una palabra en L y damos su descomposición como sigue: $z = uvwxy$ con $vx \neq \varepsilon$ y $|vwx| \leq n$. Analicemos la cadena z:

- Como $|vwx| \leq n$ entonces en vwx **no** pueden figurar los tres terminales a, b, c simultáneamente.
- Por otro lado como $v \neq \varepsilon$ ó $x \neq \varepsilon$ se distinguen dos casos:
  - En alguna de v ó x figuran dos tipos de terminales.
    En tal caso en uv^2wx^2y figuran algunas b's seguidas de a's ó algunas c's seguidas de b's rompiendo con la forma de las cadenas en L.
  - Cada una de las cadenas v y x contiene sólo un tipo de terminal. Dado que en vwx no figuran los tres terminales, en la cadena uv^2wx^2y se altera el número de dos de los terminales pero nunca de los tres, por lo que uv^2wx^2y no está en L.

De cualquier forma al "bombear" la cadena, el resultado no es parte del lenguaje. Por lo tanto L no es libre de contexto.