Modelos de computación (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 5

José Carlos Martínez Velázquez

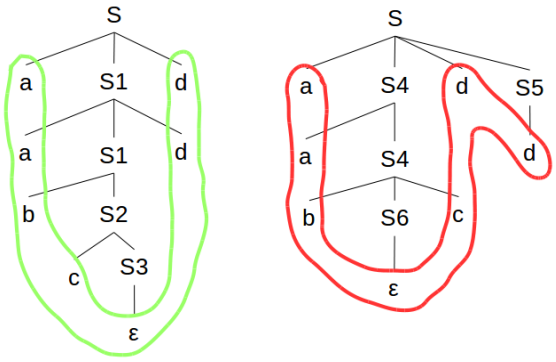
14 de enero de 2016

1. Dada la gramática:

$$
\begin{array}{ll}
S \to aS_1d & S \to aS_4dS_5 \\
S_1 \to aS_1d \mid bS_2 & S_4 \to aS_4 \mid bS_6c \\
S_2 \to bS_2 \mid cS_3 & S_5 \to dS_3 \mid d \\
S_3 \to cS_3 \mid \varepsilon & S_6 \to bS_6c \mid \varepsilon
\end{array}
$$

A. Demuestra que es ambigua.

Para demostrar que ésta gramática es ambigua, basta con encontrar una palabra que tenga dos árboles de derivación distintos. Es sencillo encontrarlos para la palabra *aabcdd*:



En color verde, se obtiene la palabra mediante un árbol de derivacion usando las reglas de producción S_1, S_2 y S_3. En color rojo, se obtiene la palabra usando las reglas de producción S_4, S_5 y S_6, todas a partir de S. Con esta búsqueda, hemos concluido que la gramática dada es ambigua.

B. Determina el lenguaje que genera la gramática.

Tal y como está dada la gramática, se puede ver que las reglas de producción S_1, S_2 y S_3 nos generan el lenguaje en que cada palabra tiene el mismo numero de a's que de d's y un número arbitrario de b's y de c's en medio (el numero de b's puede ser distinto del número de c's). Las c's van siempre precedidas por b's y todos los símbolos deben aparecer una sola vez.

Por otro lado, las reglas de producción S_4, S_5 y S_6 (también a partir de S) generan el lenguaje que tiene un número arbitrario de a's, seguido de un número de b's que es igual al número de c's y un número arbitrario de d's (el número de a's puede ser distinto del número de d's).

Aunando ambas partes, tenemos el siguiente lenguaje:

$$L = \{a^i b^j c^k d^i \ \vee \ a^i b^j c^j d^k \ \in \ \{a,b,c,d\}^* \ / \ i,j,k \geq 1\}$$

C. Encuentra una gramática no ambigua que genere el mismo lenguaje.

Para encontrar una gramática no ambigua que genere el mismo lenguaje no hay una forma de prodecer genérica. Es necesario saber por qué se está produciendo la ambigüedad y ver si es posible resolverla en cada caso.

En este caso, podemos ver que la gramática es ambigua porque hay palabras que pueden ser generadas por las reglas de producción $S_1,\ S_2,\ S_3$ o por las reglas de producción $S_4,\ S_5,\ S_6$. Cada uno de éstos bloques nos permite fijar dos letras iguales (a y d ó b y c) y las dos restantes, un número arbitrario para cada una. Ambos bloques generan un conjunto de palabras comunes, éstas son las que hacen que las letras cuyo número de apariciones puedan no ser iguales, lo sean.

Formalmente, podemos descomponer el lenguaje en dos sublenguajes:

$$L_1 = \{a^i b^j c^k d^i \ \in \ \{a,b,c,d\}^* \ / \ i,j,k \geq 1\}$$

$$L_2 = \{a^i b^j c^j d^k \ \in \ \{a,b,c,d\}^* \ / \ i,j,k \geq 1\}$$

L_1 es el lenguaje generado por las reglas de producción $S_1,\ S_2$ y S_3 y L_2 es el lenguaje generado por las reglas de producción $S_4,\ S_5$ y S_6. Si en L_1 hacemos que j=k y en L_2 hacemos que i=k, entonces vemos como L_1 y L_2 pueden generar un conjunto de palabras comunes. Dicho de otro modo, la ambigüedad se da porque $L_1 \cap L_2 \neq \emptyset$.

La forma de arreglar la ambigüedad que tiene ésta gramática es aislar los casos en que i=k e j=k en una sóla gramática y luego volver a juntar la gramática. Dicho así, parece complicado, pero veremos que no.

Descompondremos la gramática original en cuatro más pequeñas, que aislarán los casos:

- Número de a's = número de d's y número de b's >número de c's.

$$\begin{array}{ll} T \to aT_1d & T_2 \to bT_2T_3 \mid b \\ T_1 \to aT_1d \mid bT_2T_3 & T_3 \to c \mid \varepsilon \end{array}$$

- Número de a's = número de d's y número de b's <número de c's.

$$\begin{array}{ll} U \to aU_1d & U_2 \to b \mid \varepsilon \\ U_1 \to aU_1d \mid U_2U_3c & U_3 \to U_2U_3c \mid c \end{array}$$

- Número de b's = número de c's y número de a's $\geq$ número de d's (aisla los casos j=k, i=k, incluso i=j=k).

$$\begin{array}{ll} V \to aV_1V_3 & V_2 \to bV_2c \mid \varepsilon \\ V_1 \to aV_1V_3 \mid V_2 & V_3 \to d \mid \varepsilon \end{array}$$

- Número de b's = número de c's y número de a's <número de d's.

$$\begin{array}{ll} W \to aW_1dW_2 & W_2 \to bW_2c \mid \varepsilon \\ W_1 \to aW_1d \mid W_2 & W_3 \to dW_3 \mid d \end{array}$$

Una vez aislados todos los casos, ya queda reconstruir la gramática, que no es más que poner una regla de producción "padre" que "enganche" con uno y sólo un bloque al ser sustituido. Entonces, la gramática quedaría del siguiente modo:

$$S \to T \mid U \mid V \mid W$$

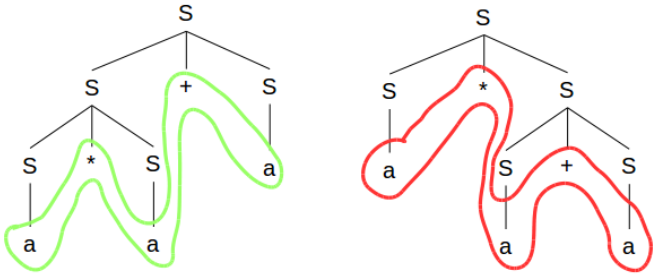
$T \to aT_1d$	$U \to aU_1d$	$V \to aV_1V_3$	$W \to aW_1dW_2$
$T_1 \to aT_1d \mid bT_2T_3$	$U_1 \to aU_1d \mid U_2U_3c$	$V_1 \to aV_1V_3 \mid V_2$	$W_1 \to aW_1d \mid W_2$
$T_2 \to bT_2T_3 \mid b$	$U_2 \to b \mid \varepsilon$	$V_2 \to bV_2c \mid \varepsilon$	$W_2 \to bW_2c \mid \varepsilon$
$T_3 \to c \mid \varepsilon$	$U_3 \to U_2U_3c \mid c$	$V_3 \to d \mid \varepsilon$	$W_3 \to dW_3 \mid d$

2. Dada la gramática:

$$\begin{array}{ll} S \to S + S & S \to (S) \\ S \to S * S & S \to a \end{array}$$

A. Demuestra que es ambigua.

En este ejercicio, al igual que en el anterior, para demostrar que la gramática es ambigua basta con encontrar dos árboles de derivación distintos para una misma palabra, en este caso $a * a + a$



B. ¿Eres capaz de encontrar una gramática que genere el mismo lenguaje y que sea no ambigua?.

La ambigüedad en este caso se da porque no esta definida la prioridad de los signos. Intentamos definirla con variables intermedias. Definiremos un árbol de prioridad de tres niveles, donde lo primero que se define es lo último que se hace. Lógicamente, en el nivel

más alto definiremos la suma, en el segundo nivel definiremos las multiplicaciones y en el último nivel definiremos los paréntesis y el símbolo terminal a. Lo normal es que dentro de los paréntesis pueda aparecer cualquier cosa, luego dentro del paréntesis volverá a aparecer el nivel más alto. La gramática resultante sería la siguiente.

$$\begin{aligned} &S \to S_1 \mid S_2 \mid S_3 \\ &S_1 \to S_2 + S_2 \mid S_2 \\ &S_2 \to S_3 * S_3 \mid S_3 \\ &S_3 \to (S) \mid a \end{aligned}$$

Ahora la palabra en la que antes encontrabamos ambigüedad, sólo tiene un árbol de derivación.

3. Dada la siguiente gramática libre de contexto:

$$\begin{array}{ll} S \to A \mid BCa \mid aDcd \mid EDF & C \to Cc \mid Bb \mid AaE \mid c \\ A \to aAb \mid c & D \to aDd \mid Dd \mid \varepsilon \\ B \to CD \mid ECd \mid Ad \mid \varepsilon & E \to aaEB \mid EFG \end{array}$$

A. Elimina las producciones inútiles

Para facilitar las cosas, reordenaremos la gramática de forma que las reglas de producción no tengan "or", y se puedan visualizar más claramente.

$$\begin{array}{llll} S \to A & B \to CD & C \to AaE & E \to EFG \\ S \to BCa & B \to ECd & C \to c & \\ S \to aDcd & B \to Ad & D \to aDd & \\ S \to EDF & B \to \varepsilon & D \to Dd & \\ A \to aAb & C \to Cc & D \to \varepsilon & \\ A \to c & C \to Bb & E \to aaEB & \end{array}$$

El algoritmo para eliminar las producciones inútiles tiene dos partes:

Primera parte:

Seleccionar aquéllas reglas de producción que generan símbolos terminales y meter la variable de la izquierda en el vector V_t, un vector de variables que llegan a símbolos terminales.

$$\begin{array}{llll} S \to A & B \to CD & C \to AaE & E \to EFG \\ S \to BCa & B \to ECd & C \to c & \\ S \to aDcd & B \to Ad & D \to aDd & \\ S \to EDF & B \to \varepsilon & D \to Dd & \\ A \to aAb & C \to Cc & D \to \varepsilon & \\ A \to c & C \to Bb & E \to aaEB & \end{array}$$

$V_t = \{A, B, C, D\}$

Ahora, por cada producción que no genera un símbolo terminal (las que no están marcadas en rojo), hay que mirar si todas sus variables (letras mayúsculas) pertenecen a V_t. Fijémonos que hay varias reglas de producción para un mismo símbolo (por ejemplo S tiene cuatro reglas de producción), con que una regla de producción cumpla que todas las variables de la parte derecha estén en V_t, vale para meter la variable de la parte izquierda en V_t.

Por cada vez que cambie V_t hay que mirar de nuevo todas las producciones que van quedando, es decir, dar vueltas revisando las reglas sin marcar, siempre que se introduce una nueva variable en V_t.

$S \to A$	$B \to CD$	$C \to AaE$	$E \to EFG$
$S \to BCa$	$B \to ECd$	$C \to c$	
$S \to aDcd$	$B \to Ad$	$D \to aDd$	
$S \to EDF$	$B \to \varepsilon$	$D \to Dd$	
$A \to aAb$	$C \to Cc$	$D \to \varepsilon$	
$A \to c$	$C \to Bb$	$E \to aaEB$	

$V_t = \{A, B, C, D, S\}$

Una vez que se termina una vuelta sin que cambie V_t, hay que eliminar todas las reglas de producción en las que aparezca una variable que no aparece en V_t.

$S \to A$	$B \to CD$	~~$C \to AaE$~~	~~$E \to EFG$~~
$S \to BCa$	~~$B \to ECd$~~	$C \to c$	
$S \to aDcd$	$B \to Ad$	$D \to aDd$	
~~$S \to EDF$~~	$B \to \varepsilon$	$D \to Dd$	
$A \to aAb$	$C \to Cc$	$D \to \varepsilon$	
$A \to c$	$C \to Bb$	~~$E \to aaEB$~~	

$V_t = \{A, B, C, D, S\}$

Cabe destacar que si S no estuviera en V_t, podrían eliminarse todas las reglas de producción de inicio y, por tanto, el lenguage generado por la gramática sería el vacío: $L = \emptyset$.

Segunda parte:

Ahora haremos una búsqueda recursiva analizando todas las variables y los símbolos terminales que se alcanzan. Definiremos tres vectores:

- J: Variables que quedan por analizar
- V_s: Variables obtenidas
- T_s: Símbolos terminales obtenidos

Inicialmente, $J = \{S\}$, $V_s = \{S\}$, $T_s = \emptyset$

$$
\begin{array}{lll}
S \to A & B \to CD & \\
S \to BCa & & C \to c \\
S \to aDcd & B \to Ad & D \to aDd \\
 & B \to \varepsilon & D \to Dd \\
A \to aAb & C \to Cc & D \to \varepsilon \\
A \to c & C \to Bb &
\end{array}
$$

Mientras que $J \neq \emptyset$ se extrae la primera variable de J y se analiza. Analizar una variable consiste en evaluar todas las reglas de producción en las que dicha variable aparece a la izquierda de la flecha y, por cada variable que aparezca a la derecha de la flecha en dicha regla de producción, mirar si está analizada (es decir, si está en V_s). Si no lo está, es que no ha sido analizada, entonces deberá introducirse en J y en V_s. Además, los símbolos terminales a la derecha de la flecha en la regla de producción deberán introducirse en T_s.

Una vez terminados todos los análisis, deberán eliminarse primero todas las reglas de producción en las que aparezcan en la parte izquierda una variable que no aparezca en V_s. Posteriormente habrá que eliminar todas las reglas de producción en las que aparezca cualquier variable de las eliminadas anteriormente (que no aparezca en V_s) o en las que aparezca cualquier variable que no aparezca en T_s

Las iteraciones, dada la gramática resultante, serían:

$$
\begin{array}{lll}
1.\ J = \{S\} & V_s = \{S\} & T_s = \emptyset \\
2.\ J = \{A, B, C, D\} & V_s = \{S, A, B, C, D\} & T_s = \{a, c, d\} \\
3.\ J = \{B, C, D\} & V_s = \{S, A, B, C, D\} & T_s = \{a, c, d, b\} \\
4.\ J = \{C, D\} & V_s = \{S, A, B, C, D\} & T_s = \{a, c, d, b, \varepsilon\} \\
5.\ J = \{D\} & V_s = \{S, A, B, C, D\} & T_s = \{a, c, d, b, \varepsilon\} \\
6.\ J = \emptyset & V_s = \{S, A, B, C, D\} & T_s = \{a, c, d, b, \varepsilon\}
\end{array}
$$

Una vez que $J = \emptyset$ el algoritmo ha finalizado. En V_s están todas las variables que aparecen en la parte izquierda de cada regla de producción. En T_s aparecen todos los símbolos terminales que aparecen en las reglas de producción, por lo que no se puede eliminar ninguna regla de producción y la gramática quedaría igual.

La gramática resultante es:

$$
\begin{array}{ll}
S \to A & B \to \varepsilon \\
S \to BCa & C \to Cc \\
S \to aDcd & C \to Bb \\
A \to aAb & C \to c \\
A \to c & D \to aDd \\
B \to CD & D \to Dd \\
B \to Ad & D \to \varepsilon
\end{array}
$$

B. Elimina las producciones nulas

Partimos de la gramática resultante de eliminar las producciones inútiles. Una producción inútil tiene la forma $A \to \varepsilon$. De cada regla de producción con ésta forma, introdu-

ciremos la variable de la izquierda de la flecha en un vector H, que contiene todas las variables anulables.

$$\begin{array}{ll} S \to A & B \to \varepsilon \\ S \to BCa & C \to Cc \\ S \to aDcd & C \to Bb \\ A \to aAb & C \to c \\ A \to c & D \to aDd \\ B \to CD & D \to Dd \\ B \to Ad & D \to \varepsilon \end{array}$$

$H = \{B, D\}$

Una vez que tenemos todas las variables anulables, debemos revisar todas las reglas de producción. Para eliminar las producciones inútiles, por cada variable anulable que aparezca a la derecha, habrá que añadir una regla de producción sin ella. Habría que "tapar con el dedo" cada variable anulable y añadir la regla de producción que resulta. Después habría que taparlas en combinaciones de dos, de tres, etc. Así hasta cubrir todas las posibles combinaciones. A continuación vemos un ejemplo.

Ejemplo 1. *Imaginemos que $H = \{A, B\}$ y tenemos una regla de producción $C \to AB$. Tapamos con el dedo A por estar en H y hay que añadir la regla de producción $C \to B$. Tapamos B por aparecer en H y añadiríamos la regla de producción $C \to A$. Hemos terminado con las combinaciones de una variable. Tapamos A y B por aparecer en H y deberíamos añadir la regla $C \to \varepsilon$ (que habría que eliminar posteriormente). Ahora habríamos terminado con las combinaciones de dos y ya no hay más combinaciones posibles.*

Marcamos aquellas que van a derivarnos en nuevas producciones y las añadimos.

$$\begin{array}{ll} S \to A & \cancel{B \to \varepsilon} \\ S \to BCa \Rightarrow S \to Ca & C \to Cc \\ S \to aDcd \Rightarrow S \to acd & C \to Bb \Rightarrow C \to b \\ A \to aAb & C \to c \\ A \to c & D \to aDd \Rightarrow D \to ad \\ B \to CD \Rightarrow B \to C & D \to Dd \Rightarrow D \to d \\ B \to Ad & \cancel{D \to \varepsilon} \end{array}$$

$H = \{B, D\}$

Cabe destacar que si $S \in H$, entonces S es anulable y la palabra vacía puede generarse con esta gramática.

La gramática resultante de eliminar las producciones nulas esla siguiente:

$S \to A$	$A \to c$	$C \to b$
$S \to BCa$	$B \to CD$	$C \to c$
$S \to Ca$	$B \to C$	$D \to aDd$
$S \to aDcd$	$B \to Ad$	$D \to ad$
$S \to acd$	$C \to Cc$	$D \to Dd$
$A \to aAb$	$C \to Bb$	$D \to d$

C. Elimina las producciones unitarias

Eliminar las producciones unitarias es sencillo. Definiremos un vector H, que contendrá un par de variables. La variable de la izquierda es la que genera y la de la derecha la generada. Una producción unitaria es de la forma $A \to B$, entonces, en este caso, introduciríamos en H el par (A, B). Hay una particularidad. Si en el vector H tuvieramos dos pares $(A, B), (B, C)$ habría que incluir en H el par (A, C) por transitividad. Vamos con nuestra gramática:

$S \to A$	$A \to c$	$C \to b$
$S \to BCa$	$B \to CD$	$C \to c$
$S \to Ca$	$B \to C$	$D \to aDd$
$S \to aDcd$	$B \to Ad$	$D \to ad$
$S \to acd$	$C \to Cc$	$D \to Dd$
$A \to aAb$	$C \to Bb$	$D \to d$

$H = \{(S, A), (B, C)\}$ En este caso no tenemos transitividad.

Por cada par que aparece en H tenemos que añadir tantas reglas de producción con la variable izquierda antes de la flecha como reglas de producción tenga la regla derecha. Veamos un ejemplo:

Ejemplo 2. *Imaginemos que $S \to A$; $A \to aB$; $A \to c$. Entonces $H = \{(S, A)\}$ y para eliminar la regla $S \to A$, tendríamos que añadir que $S \to aB$ y $S \to c$. Entonces, nuestra gramática sin producciones unitarias sería: $S \to aB$; $S \to c$; $A \to aB$; $A \to c$*

Apliquémoslo a nuestra gramática. Dado que $H = \{(S, A), (B, C)\}$, habrá que añadir las producciones siguientes:

~~$S \to A$~~	$S \to acd$	$B \to Bb$	$C \to b$
$S \to aAb$	$A \to aAb$	$B \to b$	$C \to c$
$S \to c$	$A \to c$	$B \to c$	$D \to aDd$
$S \to BCa$	$B \to CD$	$B \to Ad$	$D \to ad$
$S \to Ca$	~~$B \to C$~~	$C \to Cc$	$D \to Dd$
$S \to aDcd$	$B \to Cc$	$C \to Bb$	$D \to d$

La gramática, pues, sin producciones unitarias sería la siguiente:

$S \to aAb$	$A \to aAb$	$B \to c$	$D \to aDd$
$S \to c$	$A \to c$	$B \to Ad$	$D \to ad$
$S \to BCa$	$B \to CD$	$C \to Cc$	$D \to Dd$
$S \to Ca$	$B \to Cc$	$C \to Bb$	$D \to d$
$S \to aDcd$	$B \to Bb$	$C \to b$	
$S \to acd$	$B \to b$	$C \to c$	

D. Pasa a Forma Normal de Chomsky

Pasar a Forma Normal de Chomsky requiere en la mayoría de los casos necesita una refactorización. Una regla de producción en FNC tiene la forma $A \to BC$ con $B, C \in V$ (B y C son variables, deben aparecer dos variables) o bien $A \to a$ con $a \in T$ (en la parte derecha solo debe aparecer un símbolo terminal). Un ejemplo de refactorización es el siguiente:

Ejemplo 3. *Imaginemos que tenemos una regla de producción $S \to Abc$. Dado que A es una variable, deberíamos transformar esta regla de producción en $S \to AS_{bc}$. La regla de producción ahora está en FNC, pero no podemos dejarlo así, hay que definir qué es S_{bc}, que también debe estar en FNC. Para ello añadimos la regla de producción $S_{bc} \to S_bS_c$, que también estaría en FNC. Por último, añadimos las reglas de producción $S_b \to b$ y $S_c \to c$ (también están en FNC). Ya habríamos terminado de convertir una sola regla de producción.*

Gramática sin producciones inútiles, nulas ni unitarias:

1. $S \to aAb$	7. $A \to aAb$	13. $B \to c$	19. $D \to aDd$
2. $S \to c$	8. $A \to c$	14. $B \to Ad$	20. $D \to ad$
3. $S \to BCa$	9. $B \to CD$	15. $C \to Cc$	21. $D \to Dd$
4. $S \to Ca$	10. $B \to Cc$	16. $C \to Bb$	22. $D \to d$
5. $S \to aDcd$	11. $B \to Bb$	17. $C \to b$	
6. $S \to acd$	12. $B \to b$	18. $C \to c$	

La gramática en FNC resultante:

1. $S \to S_{bc}S_a$	$S_{Ab} \to AS_b$	15. $C \to CS_c$
$S_{bc} \to BC$	$S_b \to b$	16. $C \to BS_b$
$S_a \to a$	6. $S \to c$	17. $C \to b$
2. $S \to CS_a$	7. $A \to S_aS_{Ab}$	18. $C \to c$
3. $S \to S_{aD}S_{cd}$	8. $A \to c$	19. $D \to S_aS_{Dd}$
$S_{aD} \to S_aD$	9. $B \to CD$	$S_{Dd} \to DS_d$
$S_{cd} \to S_cSd$	10. $B \to CS_c$	20. $D \to S_aS_d$
$S_c \to c$	11. $B \to BS_b$	21. $D \to DS_d$
$S_d \to d$	12. $B \to b$	22. $D \to d$
4. $S \to S_aS_{cd}$	13. $B \to c$	
5. $S \to S_aS_{Ab}$	14. $B \to AS_d$	

Si la palabra no es generada, antes había que buscar en el nivel de profundidad 2n-1, en FNC basta con generar hasta la profundidad n para descartar que una palabra pueda ser generada.