Temas finales de
Teoría de Autómatas y Lenguajes Formales
Curso 2002-2003

M. Luisa González Díaz
Departamento de Informática
Universidad de Valladolid

1. Lenguajes independientes de contexto y autómatas con pila

1.1. Lenguajes independientes de contexto

1.1.1. Gramáticas de tipo 2 vs gramáticas independientes de contexto

Toda gramática independiente de contexto genera un lenguaje de tipo 2 (gracias al algoritmo de eliminación de reglas λ).

1.1.2. Árbol de derivación

Dada una gramática independiente de contexto G, y una cadena $\alpha \in (\Sigma_T \cup \Sigma_A)^*$ son equivalentes

1. x es una forma sentencial de G
2. Existe una derivación $S \Rightarrow^* \alpha$
3. Existe una derivación más a la derecha $S \Rightarrow^*_{m.d.} \alpha$
4. Existe una derivación más a la izquierda $S \Rightarrow^*_{m.i.} \alpha$
5. Existe un árbol de derivación en G de resultado α

En particular, si $x \in \Sigma_t^*$ son equivalentes

1. $x \in L(G)$
2. Existe una derivación $S \Rightarrow^+ x$
3. Existe una derivación más a la derecha $S \Rightarrow^+_{m.d.} x$
4. Existe una derivación más a la izquierda $S \Rightarrow^+_{m.i.} x$
5. Existe un árbol de derivación en G de resultado x

Cada árbol de derivación puede describirse mediante (posiblemente) varias derivaciones.

Cada árbol de derivación puede describirse mediante exactamente una derivación más a la derecha.

Cada árbol de derivación puede describirse mediante exactamente una derivación más a la izquierda.

Cada derivación más a la derecha describe exactamente un árbol de derivación. Cada derivación más a la izquierda describe exactamente un árbol de derivación.

1.1.3. Ambigüedad

Definición 1.1.3.1 *Dada una gramática G independiente de contexto y una forma sentencial α (respectivamente sentencia x), α es* **ambigua** *para G si existe más de un árbol de derivación en G de resultado α (respectivamente x).*

Son equivalentes:

1. α es ambigua para G
2. Existen (al menos) dos árboles de derivación distintos con resultado α
3. Existen (al menos) dos derivaciones más a la derecha con resultado α
4. Existen (al menos) dos derivaciones más a la izquierda con resultado α

Definición 1.1.3.2 *Una gramática independiente de contexto G es* **ambigua** *si existe alguna forma sentencial α ambigua para G.*

Todo lenguaje que pueda generarse mediante una gramática i.c. no ambigua, puede también ser generado mediante una gramática i.c. ambigua.

Definición 1.1.3.3 *Un lenguaje L es* **inherentemente ambiguo** *si toda gramática capaz de generarlo es ambigua.*

Proposición 1.1.3.4 *Existen lenguajes inherentemente ambiguos.*

1.1.4. Simplificación de gramáticas

Algoritmos de eliminación de símbolos y reglas inútiles, eliminación de reglas $A \to A$, eliminación de reglas λ, eliminación de reglas simples.

1.1.5. Forma normal de Chomsky

Definición 1.1.5.1 (FNC) *Una gramática está en* **forma normal de Chomsky (FNC)** *si todas sus reglas están en $\Sigma_A \times \Sigma_A\Sigma_A \cup \Sigma_A \times \Sigma_T$ (es decir, son de la forma $A_i \to A_jA_k$ ó $A_i \to a$)*

Proposición 1.1.5.2 *Dada una gramática independiente de contexto que no genere la cadena vacía, existe una gramática equivalente en FNC.*

Lema 1.1.5.3 *Sea G una gramática en FNC. Si $A \Rightarrow^* x$ mediante un árbol de profundidad p, entonces $p \leq |x| \leq 2^{p-1}$*

1.1.6. Lema de bombeo

Lema 1.1.6.1 (de bombeo para l.i.c) *Sea L un lenguaje independiente de contexto sobre un alfabeto Σ_T. Existe un número entero $N > 0$ tal que, si $z \in L$ y $|z| \geq N$ entonces existen $x, y, u, v, w \in \Sigma_T^*$ tales que $z = xyuvw$, $|yuv| \leq N$, $|yv| > 0$ y, $\forall i \geq 0$ ocurre que $xy^iuv^iw \in L$*

Este lema de bombeo permite, por ejemplo, comprobar que $\{a^nb^nc^n / n \geq 0\}$ no es i.c.

Como en el caso del lema de bombeo para lenguajes regulares, la condición no es suficiente, es decir, existen (y muchos) lenguajes que, verificando este lema de bombeo, no son independientes de contexto.

1.1.7. Propiedades de cierre de los lenguajes independientes de contexto

Proposición 1.1.7.1 (Unión) *La unión de lenguajes independientes de contexto es independiente de contexto*

Proposición 1.1.7.2 (Concatenación) *La concatenación de lenguajes independientes de contexto es independiente de contexto*

Proposición 1.1.7.3 (Cierre) *El cierre de Kleene ($*$) de un lenguaje independiente de contexto es independiente de contexto*

Proposición 1.1.7.4 (Intersección) *La intersección de lenguajes independientes de contexto* **no tiene por qué serlo**

Proposición 1.1.7.5 (Complementario) *El complementario de un lenguaje independiente de contexto* **no tiene por qué serlo**

1.1.8. Eliminación de la recursión por la izquierda

Definición 1.1.8.1 *Una regla es* **recursiva por la izquierda** *si es de la forma* $A \to A\alpha$

Se pueden eliminar las reglas recursivas por la izquierda (eliminación de la recursión directa por la izquierda)

$$\text{sustituyendo las reglas} \quad A \to A\alpha \mid \beta \quad \text{por las reglas} \quad \begin{array}{lcl} A & \to & \beta A' \\ A' & \to & \alpha A' \mid \lambda \end{array}$$

aunque a costa de introducir reglas λ

Otra posibilidad, sin introducir reglas λ:

$$\text{sustituir} \quad A \to A\alpha \mid \beta \quad \text{por} \quad \begin{array}{lcl} A & \to & \beta A' \mid \beta \\ A' & \to & \alpha A' \mid \alpha \end{array}$$

Más generalmente:

$$\text{sustituir} \quad A \to A\alpha_1 \mid A\alpha_2 \mid \ldots A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m \quad \text{por} \quad \begin{array}{lcl} A & \to & \beta_1 A' \mid \ldots \mid \beta_m A' \\ A' & \to & \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \lambda \end{array}$$

o

$$\text{sustituir} \quad A \to A\alpha_1 \mid A\alpha_2 \mid \ldots A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m \quad \text{por} \quad \begin{array}{lcl} A & \to & \beta_1 A' \mid \ldots \mid \beta_m A' \mid \beta_1 \mid \ldots \mid \beta_m \\ A' & \to & \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \mid \end{array}$$

Definición 1.1.8.2 *Una gramática es* **recursiva por la izquierda** *si existe una derivación de la forma* $A \Rightarrow^+ A\alpha$

Ejemplo:

$$\begin{array}{lcl} A & \to & Ba \\ B & \to & Cb \\ C & \to & AAb \mid a \end{array}$$

Eliminación de la recursión por la izquierda general:

Algoritmo
Entrada: una gramática i.c. sin ciclos ni reglas λ, recursiva por la izquierda
Salida: una gramática i.c. equivalente sin recursión por la izquierda
Inicio

1. Ordenar los auxiliares (como se quiera): $A_1, A_2 \ldots A_n$

2. **para** i desde 1 hasta n **hacer** (fijado un antecedente A_i)
   para j desde 1 hasta $i-1$ **hacer** (auxiliares estrictamente anteriores al fijado)
      sustituir cada regla $A_i \to A_j\gamma$
      por $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots$
      siendo $A_j \to \delta_1 \mid \delta_2 \mid \ldots$ las reglas actuales para A_j
   fin para
   (todas los consecuentes de reglas para A_i
   comienzan por un terminal o un auxiliar posterior)
   eliminar la recursión (si la hay) directa para A_i

fin para
Fin

1.1.9. Forma normal de Greibach

(No se estudiará este curso)

Definición 1.1.9.1 (FNG) *Una gramática está en* **forma normal de Greibach (FNG)** *si todas sus reglas están en* $\Sigma_A \times \Sigma_T\Sigma_A^*$ *(es decir, son de la forma* $A_i \to aA_{j_1}A_{j_2}\ldots A_{j_p}$ *ó* $A_i \to a$*)*

Proposición 1.1.9.2 *Dada una gramática independiente de contexto que no genere la cadena vacía, existe una gramática equivalente en FNG.*

1.2. Autómatas con pila

1.2.1. Autómatas con pila no deterministas

Son autómatas (alfabeto de entrada Σ_T, conjunto finito de estados $Q\ldots$) que disponen además de una pila, en la que se pueden almacenar símbolos de determinado alfabeto Γ (que puede contener a Σ_T), y cuyas transiciones se efectúan en función, no sólo del símbolo de entrada, sino también del de la cima de la pila; y al efectuar las transiciones, además de cambiar de estado, sustituyen el símbolo de la cima de la pila por una secuencia de símbolos o incluso por la cadena vacía.

Formalmente: $(\Sigma_T, \Gamma, Q, A_0, q_0, f, F)$ donde

Σ_T es un alfabeto "de entrada"

Γ es un alfabeto "de la pila"

Q es un conjunto finito "de estados"

$A_0 \in \Gamma$, "símbolo inicial de la pila"

$q_0 \in Q$, "estado inicial"

$F \subseteq Q$, conjunto de estados "finales"

f es la "función de transición"

$$f : Q \times (\Sigma_T \cup \{\lambda\}) \times \Gamma \to 2^{Q\times\Gamma^*} = \mathcal{P}(Q \times \Gamma^*)$$

que por lo tanto asocia a cada estado q de Q, símbolo terminal a de Σ_T y A de Γ, un conjunto $f(q, a, A)$ (posiblemente vacío) de pares (q', Z) de $Q\times\Gamma^*$, y también hace esta asociación para (q, λ, A)

El funcionamiento de una autómata a pila es como sigue: inicialmente se comienza en el estado q_0 y se coloca en la pila el símbolo inicial A_0. Se considera el primer símbolo de la entrada a y se revisan $f(q_0, a, A_0)$ y $f(q_0, \lambda, A_0)$. El autómata entonces sigue alguna de las instrucciones posibles que estos conjuntos indiquen. Si ambos conjuntos son vacíos, el autómata no puede moverse. Si no, y decide elegir uno de los (q_r, Z_r) de $f(q_0, a, A_0)$, entonces

1. saca A_0 de la pila
2. pone en su lugar la cadena Z_r
3. pasa al estado q_r
4. avanza en la cadena de entrada para leer el siguiente símbolo

Si decide por el contrario elegir uno de los (q_r, Z_r) de $f(q_0, \lambda, A_0)$, entonces el comportamiento es el mismo, salvo el avance en la cadena de entrada.

A partir de este momento, el autómata continúa de la misma manera, accediendo cada vez al símbolo de la cima de la pila, y al símbolo de entrada.

Para expresar en qué situación se encuentra un A.P. en un momento dado bastará con conocer su *configuración* $(q, az, AZ) \in Q \times \Sigma_T^* \times \Gamma^*$, que indica:

1. el estado q
2. la entrada restante por leer, con el primer símbolo a y el resto (z)
3. el contenido completo de la pila, con el símbolo de la cima A y la cadena de símbolos que esté debajo Z

La elección de la transición $(q_1, Z_1) \in f(q, a, A)$ hará que el A.P. pase a la configuración (q_1, z, Z_1Z), que se indicará

$$(q, az, AZ) \vdash (q_1, z, Z_1Z)$$

La elección de la transición $(q_1, Z_1) \in f(q, \lambda, A)$ hará que el A.P. pase a la configuración (q_1, az, Z_1Z), que se indicará

$$(q, az, AZ) \vdash (q_1, az, Z_1Z)$$

Según la transición que elija, el A.P. consume o no símbolo de entrada. Siempre desapila el símbolo de la cima de la pila, pero puede volverlo a apilar ($f(q, a, A) = \{(q', A)\ldots\}$, sustituirlo por una serie de símbolos ($f(q, a, A) = \{(q', A_1A_2\ldots)\ldots\}$) o solamente desapilarlo ($f(q, a, A) = \{(q', \lambda)\ldots\}$).

La posibilidad de elección entre las transiciones posibles es lo que le hace determinista o no.

Para que se considere que es determinista tiene que ocurrir:

1. $\forall (q, x, A) \in Q \times (\Sigma_T \cup \{\lambda\}) \times \Gamma$, $|f(q, x, A)| < 2$, es decir: el conjunto de transiciones posibles para cada situación se reduce a un solo elemento, o es vacío.
2. si $f(q, \lambda, A) \neq \emptyset$ entonces $f(q, a, A) = \emptyset$, $\forall a \in \Sigma_T$, es decir, si se puede hacer una transición desde un estado y con un símbolo concreto en la cima de la pila sin consumir entrada, entonces no se puede hacer ninguna transición consumiéndola.

Es decir, dada una configuración, es posible a lo sumo una configuración siguiente.

Se dice que un A.P. *acepta* una cadena *por estado final* si es posible que una sucesión de movimientos lleve al A.P. a una configuración del tipo (q_f, λ, Z) con $q_f \in F$, es decir, si consigue llegar a un estado final cuando ha leído toda la cadena de entrada (independientemente de lo que quede en la pila).

Se dice que un A.P. *acepta* una cadena *por vaciado de pila* si es posible que una sucesión de movimientos lleve al A.P. a una configuración del tipo (q, λ, λ), es decir, si consigue vaciar completamente la pila cuando ha leído toda la cadena de entrada (independientemente del que estado al que llegue sea o no final).

Por lo tanto, un AP reconoce dos lenguajes:

$$LF(A) = \{x \in \Sigma_T^* / (q_0, x, A_0) \vdash^* (q, \lambda, Z), q \in F, Z \in \Gamma^*\}$$

$$LV(A) = \{x \in \Sigma_T^* / (q_0, x, A_0) \vdash^* (q, \lambda, \lambda), q \in Q\}$$

no necesariamente iguales, y cualquiera de ellos puede ser vacío.

1.2.2. Ejemplos de autómatas con pila

$$AP_1 = (\Sigma_T = \{a, b, c\}, \Gamma = \{A, 0, 1\}, Q = \{p, q\}, A, p, f, F)$$

p	a	b	c
A	$p, 0A$	$p, 1A$	q, A
0	$p, 00$	$p, 10$	$q, 0$
1	$p, 01$	$p, 11$	$q, 1$

q	a	b	c	λ
A				q, λ
0	q, λ			
1		q, λ		

Un ejemplo de funcionamiento:

$$\begin{aligned}
(p, abcba, A) &\vdash (p, bcba, 0A) \vdash (p, cba, 10A) \vdash (q, ba, 10A) \\
&\vdash (q, a, 0A) \vdash (q, \lambda, A) \vdash (q, \lambda, \lambda) \\
(p, abcaa, A) &\vdash (p, bcaa, 0A) \vdash (p, caa, 10A) \vdash (q, aa, 10A) \\
&\quad \text{parado} \\
(p, abcbaa, A) &\vdash (p, bcbaa, 0A) \vdash (p, cbaa, 10A) \vdash (q, baa, 10A) \\
&\vdash (q, aa, 0A) \vdash (q, a, A) \vdash \text{no cambia la configuración}
\end{aligned}$$

No es difícil convencerse de que $LV(AP_1) = \{wcw^R / w \in (a|b)^*\}$. En su funcionamiento, el autómata apila los símbolos *a*, y *b* que va leyendo (recodificados como 0 y 1 respectivamente) hasta que encuentra una *c*, momento en que desaplia el símbolo de la cima con el que lee si son iguales. Sólo puede vaciar la pila si encuentra las aes y bes en orden inverso al que los leyó al principio, y sólo puede vaciarla completamente si ha encontrado una *c* (y sólo una), para, desde el estado *q*, desapilar el símbolo inicial.

El autómata es determinista. Por otra parte, si $F = \{q\}$, se tendrá que $LF(AP_1)$ es el conjunto de prefijos del lenguaje anterior, mientras que si $F = \emptyset$, entonces $LF(AP_1) = \emptyset$. Obsérvese que si $F = \{p, q\}$ no es cierto que $LF(AP_1)$ sea todo Σ_T^* (probar el comportamiento con *cc* ó *ca*).

El siguiente autómata es no determinista:

$$AP_2 = (E = \{a, b\}, \Gamma = \{A, 0, 1\}, Q = \{p, q\}, A, p, f, F)$$

p	a	b
A	$(p, 0A)$	$(p, 1A)$
0	$(p, 00); (q, \lambda)$	$(p, 10)$
1	$(p, 01)$	$(p, 11); (q, \lambda)$

q	a	b	λ
A			(q, λ)
0	(q, λ)		
1		(q, λ)	

y reconoce por vaciado de pila al lenguaje $\{ww^R / w \in (a|b)^*\}$, por un procedimiento similar al del autómata anterior. El no determinismo procede de que no es posible situar el punto medio de la cadena de entrada sin haberla leído completamente.

La pila puede "no usarse", como en el siguiente ejemplo:

$$AP_3 = (E = \{a, b\}, \Gamma = \{A\}, Q = \{p, q\}, A, p, f, F = \{q\})$$

p	a	b
A		(q, A)

q	a	b
A	(q, A)	

En este caso, el lenguaje reconoce por vaciado de pila al lenguaje vacío (nunca puede vaciarla), pero, por estado final, a ba^*. Realmente es un autómata finito, y determinista. Esta técnica puede generalizarse y demostrar que cualquier lenguaje regular puede ser reconocido por un autómata a

pila adecuado. Por lo tanto, los autómatas a pila son más potentes que los autómatas finitos, y lo son estrictamente, vistos los dos ejemplos anteriores.

El siguiente autómata es no determinista:

$$AP_4 = (E = \{a, b\}, \Gamma = \{A, a, b\}, Q = \{p, q\}, A, p, f, F = \{q\})$$

p	a	b	λ
A	(p, aA)	(p, bA)	(q, A)
a	(p, aa)	(p, λ)	
b	(p, λ)	(p, bb)	

q	a	b	λ
A			
a			
b			

y reconoce, por estado final a $\{w \in (a|b)^* / |w|_a = |w|_b\}$ y por vaciado de pila a $\emptyset$

1.2.3. Formas de reconocimiento

Un A.P. determina dos lenguajes: uno por vaciado de pila, $LV(A)$ y otro por estado final $LF(A)$. Cualquiera de ellos puede ser vacío, pueden coincidir o no. Pero se verifica el siguiente teorema, que daremos sin demostración

Teorema 1.2.3.1 *Si un AP A_1 reconoce un lenguaje L por vaciado de pila, existe otro AP A_2 que reconoce L por estado final.*

Lo único que hay que hacer es construir un A_2 que coloque su propio símbolo inicial de la pila en su pila, y encima al inicial de A_1. A partir de entonces, imita exactamente el comportamiento de A_1. Cuando éste haya vaciado su pila (lo que se detecta porque A_2 encuentra *su* símbolo inicial de pila en ella), A_2 pasa a *su* estado final.

Teorema 1.2.3.2 *Si un AP A_1 reconoce un lenguaje L por estado final, existe otro AP A_2 que reconoce L por vaciado de pila.*

Ahora un segundo autómata que imita el comportamiento del primero, cuando detecta que éste ha llegado a un estado final, pasa a un estado "de vaciado", en el que se dedica únicamente a desapilar lo que quede en la pila.

Se consideran las dos formas de reconocimiento porque, según los casos, una técnica será más directa que la otra, o más útil para lo que se desee.

La conjunción de los dos teoremas se resume en

Teorema 1.2.3.3 *El conjunto de lenguajes aceptables por vaciado de pila por el conjunto de los autómatas a pila, coincide con el conjunto de los lenguajes aceptables por estado final por dicho conjunto.*

1.2.4. Algoritmos de análisis y síntesis para lenguajes independientes de contexto

Teorema 1.2.4.1 *Dado un lenguaje independiente de contexto L, existe un A.P. cuyo lenguaje reconocido por vaciado de pila es precisamente L.*

La demostración de este teorema se basa en partir de una gramática independiente de contexto que genere L y construir a partir de ella un AP que imite, como procesos de reconocimiento, los procesos de derivación de la gramática, naturalmente en orden inverso. La construcción más sencilla puede verse en [Kelley], aunque no se exigirá este curso.

El analizador sintáctico que construye Yacc es en el fondo un AP para el caso particular de gramáticas independientes de contexto $LALR(1)$, un poco adaptado por comodidad. Las construcciones de AP para gramáticas generales independientes de contexto no podrán garantizar el determinismo (ni mucho menos el resto de propiedades que consigue Yacc).

Evidentemente, también podrá contruirse un AP que reconozca un lenguaje independiente de contexto por estado final.

Teorema 1.2.4.2 *El lenguaje reconocido por vaciado de pila por un A.P. es independiente de contexto.*

Para demostrar este teorema se construye una gramática a partir del AP que genera exactamente las cadenas que el autómata reconoce. El algoritmo genera muchos símbolos y reglas inútiles. También puede verse en [Kelley] y otros, y tampoco se exigirá este curso.

Dado que reconocer por vaciado de pila es equivalente a reconocer por estado final, también se tiene que

Teorema 1.2.4.3 *El lenguaje reconocido por estado final por un A.P. es independiente de contexto.*

En resumen, los autómatas a pila son a los lenguajes independientes de contexto como los autómatas finitos deterministas a los regulares. Sin embargo es muy importante observar que en el caso de los AP, no es equivalente el no determinismo al determinismo, sino que aquél es estrictamente más potente. Es decir, existen lenguajes independientes de contexto reconocibles por un autómata con pila no determinista que ningún AP determinista es capaz de reconocer. Por ejemplo, el del autómata AP_2 de la sección de ejemplos.

1.2.5. Propiedad adicional de los lenguajes independientes de contexto

La relación de los lenguajes independientes de contexto con los AP permite añadir la siguiente propiedad al catálogo:

Proposición 1.2.5.1 *La intersección de un lenguaje independiente de contexto y un lenguaje regular es independiente de contexto.*

Esta propiedad permite facilitar la comprobación de que un lenguaje **no** es independiente de contexto. Por ejemplo, considérese el lenguaje $L = \{a^i b^j c^k / i \neq j \vee j \neq k\}$. También puede expresarse en la forma $\{a^i b^j / i \neq j\}c^* \cup a^*\{b^j c^k / j \neq k\}$. La gramática

$$\begin{cases} S & \rightarrow & S_1 \mid S_2 \\ S_1 & \rightarrow & aS_1 b \mid A \\ S_2 & \rightarrow & aS_2 b \mid B \\ A & \rightarrow & aA \mid a \\ B & \rightarrow & bB \mid b \end{cases}$$

genera las cadenas de a^*b^* que tienen distinto número de aes que de bes, y hace a $\{a^i b^j / i \neq j\}$ independiente de contexto. Por las propiedades vistas de unión y concatenación de l.i.c, L es independiente de contexto. Sin embargo, su complementario no lo es, dado que $\overline{L} \cap a^*b^*c^* = \{a^n b^n c^n / n \geq 0\}$, que no es i.c.