Lenguajes Independientes del Contexto y Autómatas de Pila

Teoría de la Computación

Grado en Ingeniería Informática

Contenidos

Contenidos

1 Gramáticas regulares y lenguajes regulares

Nos centraremos ahora en una nueva clase de lenguajes llamados **lenguajes independientes del contexto** o **LIC,s**. Esta clase es más amplia que la de los lenguajes regulares y, de hecho, los incluye. Los LIC,s pueden especificarse mediante una notación natural y recursiva denominada **gramática independiente del contexto** o **GIC**. Los lenguajes regulares pueden denotarse también mediante un tipo de gramáticas, las gramáticas regulares, que, como veremos, son un caso particular de las GIC,s.

1 Gramáticas regulares y lenguajes regulares

Nos centraremos ahora en una nueva clase de lenguajes llamados **lenguajes independientes del contexto** o **LIC,s**. Esta clase es más amplia que la de los lenguajes regulares y, de hecho, los incluye. Los LIC,s pueden especificarse mediante una notación natural y recursiva denominada **gramática independiente del contexto** o **GIC**. Los lenguajes regulares pueden denotarse también mediante un tipo de gramáticas, las gramáticas regulares, que, como veremos, son un caso particular de las GIC,s.

Este capítulo introduce también la noción de **árbol sintáctico**, como representación gráfica de la estructura que la gramática aplica sobre las cadenas de su lenguaje. Veremos también una serie de propiedades específicas de los LIC,s entre las que destaca la posibilidad de resolver el problema de la pertenencia de una cadena de símbolos a un lenguaje de esta clase. Este problema puede abordarse mediante el diseño de **algoritmos de análisis sintáctico** o mediante el uso de los **autómatas de pila** o **AP,s**.

1 Gramáticas regulares y lenguajes regulares

Nos centraremos ahora en una nueva clase de lenguajes llamados **lenguajes independientes del contexto** o **LIC,s**. Esta clase es más amplia que la de los lenguajes regulares y, de hecho, los incluye. Los LIC,s pueden especificarse mediante una notación natural y recursiva denominada **gramática independiente del contexto** o **GIC**. Los lenguajes regulares pueden denotarse también mediante un tipo de gramáticas, las gramáticas regulares, que, como veremos, son un caso particular de las GIC,s.

Este capítulo introduce también la noción de **árbol sintáctico**, como representación gráfica de la estructura que la gramática aplica sobre las cadenas de su lenguaje. Veremos también una serie de propiedades específicas de los LIC,s entre las que destaca la posibilidad de resolver el problema de la pertenencia de una cadena de símbolos a un lenguaje de esta clase. Este problema puede abordarse mediante el diseño de **algoritmos de análisis sintáctico** o mediante el uso de los **autómatas de pila** o **AP,s**.

Desde el punto de vista práctico, las GIC,s han desempeñado un papel primordial en el desarrollo de los ordenadores desde los años sesenta. Gracias a ellas, la implementación del analizador sintáctico de un compilador dejó de ser una tarea costosa y específica. Actualmente, las GIC,s se usan para la descripción de nuevos lenguajes de programación, de formatos de documentos que aseguran intercambios de información más fiables, o incluso de algunos de los aspectos más comunes de los lenguajes naturales.

1 Gramáticas regulares y lenguajes regulares

Las ER,s y los AF,s permiten especificar lenguajes regulares. Las ER,s constituyen una plantilla o patrón generador de cadenas, mientras que los AF,s constituyen el mecanismo de reconocimiento.

1 Gramáticas regulares y lenguajes regulares

Las ER,s y los AF,s permiten especificar lenguajes regulares. Las ER,s constituyen una plantilla o patrón generador de cadenas, mientras que los AF,s constituyen el mecanismo de reconocimiento.

Sin embargo, el verdadero formalismo generador viene dado por las gramáticas regulares.

Definición

Una **gramática regular** es una colección de cuatro elementos $G = (N, \Sigma, P, S)$, donde:

- N es un conjunto finito de *símbolos no terminales* o *variables*.
- Σ es un conjunto finito de *símbolos terminales* o *alfabeto*
- P es un conjunto finito de *producciones* o *reglas de reescritura* de la forma $A \rightarrow w$, con $A \in N$ y $w \in (N \cup \Sigma)^*$, pero con las siguientes condiciones:
  - La cadena w tiene como máximo un símbolo no terminal.
  - Si w tiene un símbolo no terminal, entonces dicho símbolo es el último de la cadena.
- S es un símbolo destacado de N, denominado *símbolo inicial* o *axioma* de la gramática.

Por ejemplo, consideremos la gramática:

$$G = (N = \{S, A\},\ \Sigma = \{a, b\},\ P = \{S \to bA, A \to aaA \mid b \mid \epsilon\},\ S = S)$$

Las cadenas que forman parte del lenguaje generado por G son aquéllas que están compuestas sólo por símbolos terminales y que se pueden obtener desde S aplicando reglas de P. En este caso, se trata de cadenas que comienzan por una b, seguida de un número par de $a$es, y que terminan con ϵ o con b:

$$S \Rightarrow bA \Rightarrow baaA \Rightarrow baaaaA \Rightarrow \cdots \Rightarrow baaaa\ldots aa \quad \text{o bien} \quad baaaa\ldots aab$$

Por tanto, esto se puede escribir también como:

$$S \stackrel{*}{\Rightarrow} b(aa)^*(b \cup \epsilon)$$

1 Gramáticas regulares y lenguajes regulares

Por ejemplo, consideremos la gramática:

$$G = (N = \{S, A\},\ \Sigma = \{a, b\},\ P = \{S \to bA, A \to aaA \mid b \mid \epsilon\},\ S = S)$$

Las cadenas que forman parte del lenguaje generado por G son aquéllas que están compuestas sólo por símbolos terminales y que se pueden obtener desde S aplicando reglas de P. En este caso, se trata de cadenas que comienzan por una b, seguida de un número par de $a$es, y que terminan con ϵ o con b:

$$S \Rightarrow bA \Rightarrow baaA \Rightarrow baaaaA \Rightarrow \cdots \Rightarrow baaaa\ldots aa \quad \text{o bien} \quad baaaa\ldots aab$$

Por tanto, esto se puede escribir también como:

$$S \stackrel{*}{\Rightarrow} b(aa)^*(b \cup \epsilon)$$

Definición

El **lenguaje generado por una gramática** regular G se define entonces como:

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$$

1 Gramáticas regulares y lenguajes regulares

Teorema

Dado cualquier lenguaje regular L, existe una gramática regular G que lo genera.

Teorema

Dado cualquier lenguaje regular L, existe una gramática regular G que lo genera.

Demostración:

Si L es regular, existe un AF $M = (Q, \Sigma, s, \delta, F)$ tal que $L(M) = L$. Podemos entonces construir $G = (N, \Sigma, P, S)$, una gramática regular tal que $L(G) = L$, como sigue:

$$N = Q \qquad \Sigma = \Sigma \qquad S = s$$

$$P = \{q_i \to a\, q_j \mid \delta(q_i, a) = q_j\} \cup \{q \to \epsilon \mid q \in F\}$$

□

Teorema

Dado cualquier lenguaje regular L, existe una gramática regular G que lo genera.

Demostración:

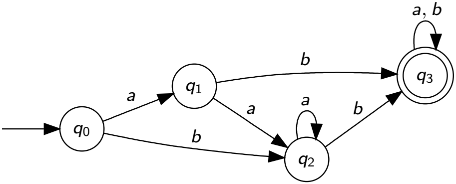
Si L es regular, existe un AF $M = (Q, \Sigma, s, \delta, F)$ tal que $L(M) = L$. Podemos entonces construir $G = (N, \Sigma, P, S)$, una gramática regular tal que $L(G) = L$, como sigue:

$$N = Q \qquad \Sigma = \Sigma \qquad S = s$$

$$P = \{q_i \to a\, q_j \mid \delta(q_i, a) = q_j\} \cup \{q \to \epsilon \mid q \in F\}$$

□

Por ejemplo, la gramática regular que genera el lenguaje aceptado por el AF de la izquierda es la que se muestra a la derecha:



$$N = \{q_0, q_1, q_2, q_3\} \quad \Sigma = \{a, b\} \quad S = q_0$$

$$\begin{aligned} P = \{\ & q_0 \to a\, q_1 \mid b\, q_2, \\ & q_1 \to a\, q_2 \mid b\, q_3, \\ & q_2 \to a\, q_2 \mid b\, q_3, \\ & q_3 \to a\, q_3 \mid b\, q_3 \mid \epsilon\ \} \end{aligned}$$

Teorema

Dada cualquier gramática regular G, existe un AFN M tal que $L(M) = L(G)$.

1 Gramáticas regulares y lenguajes regulares

Teorema

Dada cualquier gramática regular G, existe un AFN M tal que $L(M) = L(G)$.

<u>Demostración</u>:

Si $G = (N, \Sigma, P, S)$, definimos $M = (Q, \Sigma, s, \Delta, F)$, donde:

$$Q = N \cup \{f\} \cup \{\text{ estados extra para produciones con más de un terminal}\}$$

$$\Sigma = \Sigma \qquad s = S \qquad F = \{f\}$$

y por último definimos Δ a partir de las producciones de P, estudiando los casos de cada una de ellas:

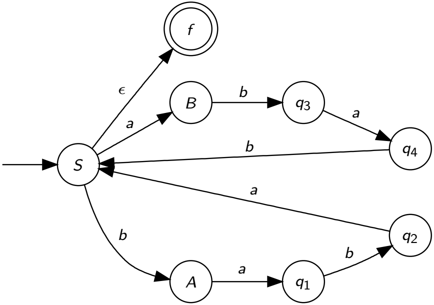
- Si $A \rightarrow \sigma_1\sigma_2 \dots \sigma_n B$
  creamos nuevos estados extra $q_1, q_2, \dots, q_{n-1}$
  y hacemos $\Delta(A, \sigma_1) = q_1$, $\Delta(q_1, \sigma_2) = q_2$, $\dots$, $\Delta(q_{n-1}, \sigma_n) = B$.
- Si $A \rightarrow \sigma_1\sigma_2 \dots \sigma_n$
  creamos nuevos estados extra $q_1, q_2, \dots, q_{n-1}$
  y hacemos $\Delta(A, \sigma_1) = q_1$, $\Delta(q_1, \sigma_2) = q_2$, $\dots$, $\Delta(q_{n-1}, \sigma_n) = f$. $\square$

Por ejemplo, el AF que acepta el leguaje generado por la gramática regular de la izquierda es el que se muestra a la derecha:

$$S \to a\,B \mid b\,A \mid \epsilon$$

$$A \to a\,b\,a\,S$$

$$B \to b\,a\,b\,S$$

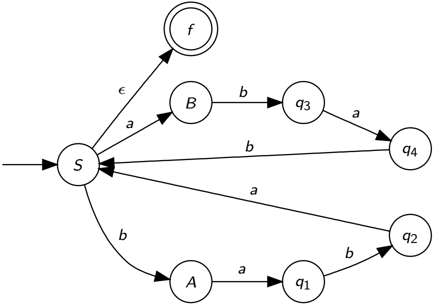


Por ejemplo, el AF que acepta el leguaje generado por la gramática regular de la izquierda es el que se muestra a la derecha:

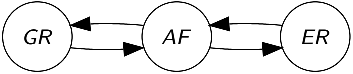
$$S \to a\,B \mid b\,A \mid \epsilon$$

$$A \to a\,b\,a\,S$$

$$B \to b\,a\,b\,S$$



En resumen, disponemos de tres mecanismos para especificar lenguajes regulares y las relaciones que hemos visto entre ellos son las siguientes:



1 Gramáticas regulares y lenguajes regulares

Definición

Una **gramática regular** se dice que es:

- o bien **lineal por la izquierda** si las reglas son de la forma $A \to w\,B$
- o bien **lineal por la derecha** si las reglas son de la forma $A \to B\,w$

donde $A, B \in N$ y $w \in \Sigma^+$.

Ambas definiciones son equivalentes, en el sentido de que se puede pasar de una a otra y viceversa.

1 Gramáticas regulares y lenguajes regulares

Definición

Una **gramática regular** se dice que es:

- o bien **lineal por la izquierda** si las reglas son de la forma $A \to w\, B$
- o bien **lineal por la derecha** si las reglas son de la forma $A \to B\, w$

donde $A, B \in N$ y $w \in \Sigma^+$.

Ambas definiciones son equivalentes, en el sentido de que se puede pasar de una a otra y viceversa.

Atención

Sin embargo, para que el lenguaje generado sea regular, estos dos formatos de reglas no se pueden mezclar entre ellos. Por ejemplo, la gramática:

$$S \to aA \mid \epsilon \qquad\qquad A \to Sb$$

puede generar la siguiente cadena:

$$S \Rightarrow aA \Rightarrow aSb \Rightarrow aaAb \Rightarrow aaSbb \Rightarrow aaaAbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

y, en general, genera $\{a^n b^n \mid n \geq 0\}$, que no es un lenguaje regular, tal y como demostramos en su momento mediante el lema del bombeo.

Contenidos

2 Gramáticas independientes del contexto

Definición

Si en una gramática $G = (N, \Sigma, P, S)$ permitimos que $P \subseteq N \times (N \cup \Sigma)^*$, es decir, que las partes derechas de las producciones puedan tener cualquier número de terminales y no terminales y en cualquier orden, la gramática se denomina **gramática independiente del contexto** o **GIC**.

2 Gramáticas independientes del contexto

Definición

Si en una gramática $G = (N, \Sigma, P, S)$ permitimos que $P \subseteq N \times (N \cup \Sigma)^*$, es decir, que las partes derechas de las producciones puedan tener cualquier número de terminales y no terminales y en cualquier orden, la gramática se denomina **gramática independiente del contexto** o **GIC**.

Definición

El lenguaje generado por una GIC G se define también como:

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$$

y se denomina **lenguaje independiente del contexto** o **LIC**.

2 Gramáticas independientes del contexto

Definición

Si en una gramática $G = (N, \Sigma, P, S)$ permitimos que $P \subseteq N \times (N \cup \Sigma)^*$, es decir, que las partes derechas de las producciones puedan tener cualquier número de terminales y no terminales y en cualquier orden, la gramática se denomina **gramática independiente del contexto** o **GIC**.

Definición

El lenguaje generado por una GIC G se define también como:

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$$

y se denomina **lenguaje independiente del contexto** o **LIC**.

Por lo tanto, toda gramática regular es también una GIC y todo lenguaje regular es también un LIC. Y además, los lenguajes regulares están incluidos de manera estricta dentro de los LIC,s ya que existen LIC,s que no son lenguajes regulares.

Por ejemplo, la gramática:

$$S \to aSb \mid \epsilon$$

genera $\{a^n b^n \mid n \geq 0\}$ que, como ya hemos visto, no es un lenguaje regular.

Ejercicio

La gramática G dada por

$$S \to aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Ejercicio

La gramática G dada por

$$S \to aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Solución:

$L(G) = ((a \cup b)(a \cup b))^*$ y G' puede venir dada por $S \to aaS \mid abS \mid baS \mid bbS \mid \epsilon$.

Ejercicio

La gramática G dada por

$$S \to aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Solución:

$L(G) = ((a \cup b)(a \cup b))^*$ y G' puede venir dada por $S \to aaS \mid abS \mid baS \mid bbS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid m \geq n\}$.

Ejercicio

La gramática G dada por

$$S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Solución:

$L(G) = ((a \cup b)(a \cup b))^*$ y G' puede venir dada por $S \rightarrow aaS \mid abS \mid baS \mid bbS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid m \geq n\}$.

Solución: $S \rightarrow aSb \mid aS \mid \epsilon$.

Ejercicio

La gramática G dada por

$$S \to aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Solución:

$L(G) = ((a \cup b)(a \cup b))^*$ y G' puede venir dada por $S \to aaS \mid abS \mid baS \mid bbS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid m \geq n\}$.

Solución: $S \to aSb \mid aS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid n \leq m \leq 2n\}$.

Ejercicio

La gramática G dada por

$$S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

no es regular, pero $L(G)$ sí es regular. Diga cuál es ese lenguaje y obtenga una gramática regular G' tal que $L(G) = L(G')$.

Solución:

$L(G) = ((a \cup b)(a \cup b))^*$ y G' puede venir dada por $S \rightarrow aaS \mid abS \mid baS \mid bbS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid m \geq n\}$.

Solución: $S \rightarrow aSb \mid aS \mid \epsilon$.

Ejercicio

Obtenga una GIC que genere $\{a^m b^n \mid n \leq m \leq 2n\}$.

Solución:

$$\begin{aligned} S &\rightarrow ASb \mid \epsilon \\ A &\rightarrow a \mid aa \end{aligned}$$

Contenidos

3 Árboles de derivación y ambigüedad

Dada la gramática:

$$1)\ S \to AB \qquad 2)\ A \to aA \qquad 3)\ A \to a \qquad 4)\ B \to bB \qquad 5)\ B \to b$$

podemos construir la cadena *aabbb* aplicando derivaciones de diferentes maneras:

$$S \overset{1}{\Longrightarrow} \underline{A}B \overset{2}{\Longrightarrow} a\underline{A}B \overset{3}{\Longrightarrow} aa\underline{B} \overset{4}{\Longrightarrow} aab\underline{B} \overset{4}{\Longrightarrow} aabb\underline{B} \overset{5}{\Longrightarrow} aabbb \qquad \text{(por la izquierda)}$$
$$S \overset{1}{\Longrightarrow} A\underline{B} \overset{4}{\Longrightarrow} Ab\underline{B} \overset{4}{\Longrightarrow} Abb\underline{B} \overset{5}{\Longrightarrow} \underline{A}bbb \overset{2}{\Longrightarrow} a\underline{A}bbb \overset{3}{\Longrightarrow} aabbb \qquad \text{(por la derecha)}$$
$$S \overset{1}{\Longrightarrow} A\underline{B} \overset{4}{\Longrightarrow} \underline{A}bB \overset{2}{\Longrightarrow} aAb\underline{B} \overset{4}{\Longrightarrow} a\underline{A}bbB \overset{3}{\Longrightarrow} aabb\underline{B} \overset{5}{\Longrightarrow} aabbb \qquad \text{(de forma alterna)}$$

3 Árboles de derivación y ambigüedad

Dada la gramática:

1) $S \to AB$ 2) $A \to aA$ 3) $A \to a$ 4) $B \to bB$ 5) $B \to b$

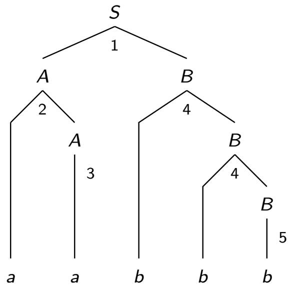
podemos construir la cadena *aabbb* aplicando derivaciones de diferentes maneras:

$$
\begin{array}{ll}
S \stackrel{1}{\Longrightarrow} \underline{A}B \stackrel{2}{\Longrightarrow} a\underline{A}B \stackrel{3}{\Longrightarrow} aa\underline{B} \stackrel{4}{\Longrightarrow} aab\underline{B} \stackrel{4}{\Longrightarrow} aabb\underline{B} \stackrel{5}{\Longrightarrow} aabbb & \text{(por la izquierda)} \\
S \stackrel{1}{\Longrightarrow} A\underline{B} \stackrel{4}{\Longrightarrow} Ab\underline{B} \stackrel{4}{\Longrightarrow} Abb\underline{B} \stackrel{5}{\Longrightarrow} \underline{A}bbb \stackrel{2}{\Longrightarrow} a\underline{A}bbb \stackrel{3}{\Longrightarrow} aabbb & \text{(por la derecha)} \\
S \stackrel{1}{\Longrightarrow} A\underline{B} \stackrel{4}{\Longrightarrow} \underline{A}bB \stackrel{2}{\Longrightarrow} aAb\underline{B} \stackrel{4}{\Longrightarrow} a\underline{A}bbB \stackrel{3}{\Longrightarrow} aabb\underline{B} \stackrel{5}{\Longrightarrow} aabbb & \text{(de forma alterna)}
\end{array}
$$

Pero todas ellas tendrán el mismo árbol de derivación.

Definición

El **árbol de derivación** de una cadena se construye creando un nodo raíz para el axioma, con tantos nodos hijos como símbolos tenga la parte derecha de la regla utilizada, y aplicando este procedimiento recursivamente sobre todos los símbolos no terminales, hasta obtener un árbol cuyas hojas sean los terminales de la cadena.



Por tanto, el árbol de derivación puede verse como la **forma canónica** de todas estas posibles derivaciones.

3 Árboles de derivación y ambigüedad

Pero esto no siempre es así. Estudiemos, por ejemplo, la cadena *abaca* con la gramática:

$$1)\ S \to SbS \qquad 2)\ S \to ScS \qquad 3)\ S \to a$$

Existen, al menos, las dos siguientes derivaciones (en este caso, ambas por la izquierda):

$$S \overset{1}{\Longrightarrow} \underline{S}bS \overset{3}{\Longrightarrow} ab\underline{S} \overset{2}{\Longrightarrow} ab\underline{S}cS \overset{3}{\Longrightarrow} abac\underline{S} \overset{3}{\Longrightarrow} abaca$$
$$S \overset{2}{\Longrightarrow} \underline{S}cS \overset{1}{\Longrightarrow} \underline{S}bScS \overset{3}{\Longrightarrow} ab\underline{S}cS \overset{3}{\Longrightarrow} abac\underline{S} \overset{3}{\Longrightarrow} abaca$$

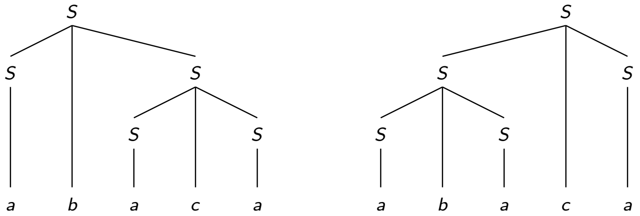
Pero esto no siempre es así. Estudiemos, por ejemplo, la cadena *abaca* con la gramática:

$$1)\ S \to SbS \qquad 2)\ S \to ScS \qquad 3)\ S \to a$$

Existen, al menos, las dos siguientes derivaciones (en este caso, ambas por la izquierda):

$$S \overset{1}{\Longrightarrow} \underline{S}bS \overset{3}{\Longrightarrow} ab\underline{S} \overset{2}{\Longrightarrow} ab\underline{S}cS \overset{3}{\Longrightarrow} abac\underline{S} \overset{3}{\Longrightarrow} abaca$$
$$S \overset{2}{\Longrightarrow} \underline{S}cS \overset{1}{\Longrightarrow} \underline{S}bScS \overset{3}{\Longrightarrow} ab\underline{S}cS \overset{3}{\Longrightarrow} abac\underline{S} \overset{3}{\Longrightarrow} abaca$$

Y cada una de ellas tiene un árbol de derivación distinto:



Definición

Si hacemos derivaciones siempre por la izquierda (o siempre por la derecha) y existen dos o más árboles de derivación distintos para una misma cadena, se dice que la gramática es **ambigua**.

3 Árboles de derivación y ambigüedad

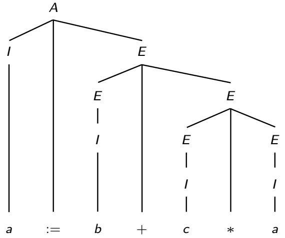
El problema de la ambigüedad es muy común en los **lenguajes naturales**. Por ejemplo, si consideramos aisladamente, es decir, fuera de todo contexto, la frase "*Veo un hombre con un telescopio*", no es posible saber si el sintagma preposicional "*con un telescopio*" está complementando al sujeto o al objeto directo de dicha oración.

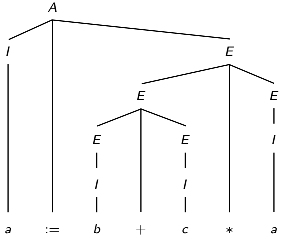
3 Árboles de derivación y ambigüedad

El problema de la ambigüedad es muy común en los **lenguajes naturales**. Por ejemplo, si consideramos aisladamente, es decir, fuera de todo contexto, la frase "*Veo un hombre con un telescopio*", no es posible saber si el sintagma preposicional "*con un telescopio*" está complementando al sujeto o al objeto directo de dicha oración.

Y la ambigüedad puede aparecer también en los **lenguajes de programación**, por ejemplo, con las estructuras `if-then-else` o con las expresiones aritméticas:

$$A \to I := E \qquad I \to a \mid b \mid c \qquad E \to E + E \mid E * E \mid (E) \mid I$$



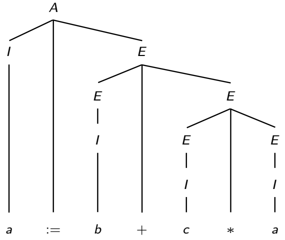


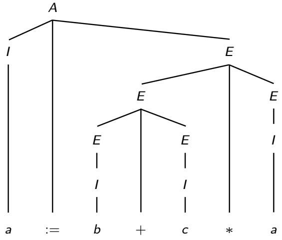
3 Árboles de derivación y ambigüedad

El problema de la ambigüedad es muy común en los **lenguajes naturales**. Por ejemplo, si consideramos aisladamente, es decir, fuera de todo contexto, la frase "*Veo un hombre con un telescopio*", no es posible saber si el sintagma preposicional "*con un telescopio*" está complementando al sujeto o al objeto directo de dicha oración.

Y la ambigüedad puede aparecer también en los **lenguajes de programación**, por ejemplo, con las estructuras `if-then-else` o con las expresiones aritméticas:

$$A \to I := E \qquad I \to a \mid b \mid c \qquad E \to E + E \mid E * E \mid (E) \mid I$$





En ambos casos es necesario definir más cosas, como por ejemplo las prioridades de los operadores, para eliminar estas ambigüedades, de tal forma que el compilador pueda siempre deducir cuál es el código ejecutable que debe generar.

Atención

¿Podemos entonces decir que un lenguaje generado por una gramática ambigua es ambiguo? La respuesta es que no. Por ejemplo, la siguiente gramática es ambigua:

$$S \to A \mid B \qquad A \to a \qquad B \to a$$

pero genera un lenguaje que puede ser generado también por la siguiente gramática no ambigua:

$$S \to a$$

Por lo tanto, en principio, **el concepto de ambigüedad se define para las gramáticas**.

Atención

¿Podemos entonces decir que un lenguaje generado por una gramática ambigua es ambiguo? La respuesta es que no. Por ejemplo, la siguiente gramática es ambigua:

$$S \to A \mid B \qquad A \to a \qquad B \to a$$

pero genera un lenguaje que puede ser generado también por la siguiente gramática no ambigua:

$$S \to a$$

Por lo tanto, en principio, **el concepto de ambigüedad se define para las gramáticas**.

Definición

No obstante, sí es cierto que existen **lenguajes inherentemente ambiguos**, es decir, que cualquier gramática que los genere será siempre ambigua.

Esto es lo que ocurre, por ejemplo, con el lenguaje $\{a^i b^j c^k \mid i = j \text{ o bien } j = k\}$.

Hay un tipo de árboles que controlan que $i = j$ con cualquier número de $c$es, y otro tipo de árboles que controlan que $j = k$ con cualquier número de $a$es. Pero cuando la cadena en cuestión verifique que $i = j = k$, ambos tipos de árboles serán válidos, lo que implica que el lenguaje es inherentemente ambiguo.

Ejercicio

Obtenga una GIC que genere $\{a^i b^j c^k \mid i = j \text{ o bien } j = k\}$.

Ejercicio

Obtenga una GIC que genere $\{a^i b^j c^k \mid i = j \text{ o bien } j = k\}$.

Solución:

$$S \to AB \mid CD \qquad A \to aAb \mid \epsilon \qquad B \to cB \mid \epsilon \qquad C \to aC \mid \epsilon \qquad D \to bDc \mid \epsilon$$

Ejercicio

Obtenga una GIC que genere $\{a^i b^j c^k \mid i = j \text{ o bien } j = k\}$.

<u>Solución</u>:

$$S \rightarrow AB \mid CD \qquad A \rightarrow aAb \mid \epsilon \qquad B \rightarrow cB \mid \epsilon \qquad C \rightarrow aC \mid \epsilon \qquad D \rightarrow bDc \mid \epsilon$$

Ejercicio

¿Existe algún lenguaje regular inherentemente ambiguo?

Ejercicio

Obtenga una GIC que genere $\{a^i b^j c^k \mid i = j \text{ o bien } j = k\}$.

<u>Solución</u>:

$$S \to AB \mid CD \qquad A \to aAb \mid \epsilon \qquad B \to cB \mid \epsilon \qquad C \to aC \mid \epsilon \qquad D \to bDc \mid \epsilon$$

Ejercicio

¿Existe algún lenguaje regular inherentemente ambiguo?

<u>Solución</u>:

La respuesta es que no. Podemos efectivamente escribir una gramática regular ambigua, como por ejemplo:

$$S \to aA \mid aB \qquad A \to a \qquad B \to a$$

Pero si obtenemos el AFN que acepta el lenguaje generado por esta gramática, lo determinizamos y obtenemos una nueva gramática a partir del correspondiente AFD, dicha gramática será siempre no ambigua.

Es decir, dado que el proceso de determinización de un AFN es siempre posible, la conversión de una gramática regular ambigua en otra no ambigua también lo es, lo cual implica la no existencia de lenguajes regulares inherentemente ambiguos.

Contenidos

4 Simplificación de gramáticas independientes del contexto

Según lo visto hasta ahora, la definición de GIC no proporciona demasiado control sobre el tipo de producciones permitidas. Así pues, introduciremos una serie de restricciones para que las producciones formen árboles de derivación que no sean innecesariamente complejos o inútilmente sencillos. Esto nos permitirá alcanzar un modelo estándar o **forma normal** para las producciones. En todo caso, debemos conseguir que estas restricciones permitan seguir generando los mismos lenguajes.

4 Simplificación de gramáticas independientes del contexto

Según lo visto hasta ahora, la definición de GIC no proporciona demasiado control sobre el tipo de producciones permitidas. Así pues, introduciremos una serie de restricciones para que las producciones formen árboles de derivación que no sean innecesariamente complejos o inútilmente sencillos. Esto nos permitirá alcanzar un modelo estándar o **forma normal** para las producciones. En todo caso, debemos conseguir que estas restricciones permitan seguir generando los mismos lenguajes.

Como primer paso, necesitamos "limpiar" las gramáticas para **eliminar símbolos y producciones inútiles**. Consideremos, por ejemplo, la siguiente gramática:

$$S \to Aa \mid B \mid D \qquad A \to aA \mid bA \mid B \qquad B \to b \qquad C \to abd$$

Se puede observar que:

- El símbolo C nunca se alcanza, por lo que podemos eliminarlo, junto con todas sus reescrituras.
- El símbolo d sólo aparece en una reescritura de C, por lo que también se puede eliminar.
- El símbolo D no se reescribe en nada, por lo que podría ser también eliminado.
- El símbolo B es, en cierto modo, redundante, por lo que podría eliminarse y cambiar $S \to B$ por $S \to b$ y $A \to B$ por $A \to b$.

4 Simplificación de gramáticas independientes del contexto

Eliminación de símbolos no terminales inútiles. Dada $G = (N, \Sigma, P, S)$ la transformamos en otra GIC $G' = (N', \Sigma, P', S)$ equivalente, es decir, tal que $L(G) = L(G')$, de forma que:

$$\forall A \in N', A \overset{*}{\Rightarrow} w, \text{para algún } w \in \Sigma^*$$

Es decir, N' contendrá sólo aquellos símbolos de N que produzcan cadenas de terminales.

Eliminación de símbolos no terminales inútiles. Dada $G = (N, \Sigma, P, S)$ la transformamos en otra GIC $G' = (N', \Sigma, P', S)$ equivalente, es decir, tal que $L(G) = L(G')$, de forma que:

$$\forall A \in N', A \overset{*}{\Rightarrow} w, \text{para algún } w \in \Sigma^*$$

Es decir, N' contendrá sólo aquellos símbolos de N que produzcan cadenas de terminales.

El algoritmo es el siguiente:

1. Inicializar N' con todos los no terminales A tales que $A \rightarrow w \in P$, con $w \in \Sigma^*$.
2. Inicializar P' con todas las producciones $A \rightarrow w$ tales que $A \in N'$ y $w \in \Sigma^*$.
3. Añadir a N' todos los no terminales A tales que $A \rightarrow w$, con $w \in (N' \cup \Sigma)^*$, es una producción de P, y añadir también dicha producción a P'.
4. Repetir el paso anterior hasta que no se puedan añadir más no terminales a N'.

4 Simplificación de gramáticas independientes del contexto

Eliminación de símbolos no terminales inútiles. Dada $G = (N, \Sigma, P, S)$ la transformamos en otra GIC $G' = (N', \Sigma, P', S)$ equivalente, es decir, tal que $L(G) = L(G')$, de forma que:

$$\forall A \in N', A \stackrel{*}{\Rightarrow} w, \text{para algún } w \in \Sigma^*$$

Es decir, N' contendrá sólo aquellos símbolos de N que produzcan cadenas de terminales.

El algoritmo es el siguiente:

1. Inicializar N' con todos los no terminales A tales que $A \rightarrow w \in P$, con $w \in \Sigma^*$.
2. Inicializar P' con todas las producciones $A \rightarrow w$ tales que $A \in N'$ y $w \in \Sigma^*$.
3. Añadir a N' todos los no terminales A tales que $A \rightarrow w$, con $w \in (N' \cup \Sigma)^*$, es una producción de P, y añadir también dicha producción a P'.
4. Repetir el paso anterior hasta que no se puedan añadir más no terminales a N'.

Si aplicamos este algoritmo a la gramática del ejemplo anterior, obtenemos:

$$S \rightarrow Aa \mid B \qquad A \rightarrow aA \mid bA \mid B \qquad B \rightarrow b \qquad C \rightarrow abd$$

donde se puede observar que hemos eliminado el símbolo D y la producción $S \rightarrow D$.

Este algoritmo termina, ya que N y P son finitos. Esencialmente, recorre hacia arriba todos los árboles de análisis, a partir de las cadenas de terminales, anotando los no terminales y las producciones que pueden producir algo. Sin embargo, la regla $C \rightarrow abd$ permanece, porque C produce terminales. Pero C no se puede alcanzar. Así pues, necesitamos un segundo paso.

Eliminación de símbolos no accesibles. Dada $G = (N, \Sigma, P, S)$ construimos una nueva GIC $G' = (N', \Sigma', P', S)$, tal que $L(G) = L(G')$, y donde:

$$\forall X \in (N' \cup \Sigma'), S \stackrel{*}{\Rightarrow} \alpha X \beta, \text{con } \alpha, \beta \in (N' \cup \Sigma')^*$$

Es decir, N' y Σ' contendrán sólo aquellos símbolos que se puedan alcanzar desde el axioma.

4 Simplificación de gramáticas independientes del contexto

Eliminación de símbolos no accesibles. Dada $G = (N, \Sigma, P, S)$ construimos una nueva GIC $G' = (N', \Sigma', P', S)$, tal que $L(G) = L(G')$, y donde:

$$\forall X \in (N' \cup \Sigma'), S \stackrel{*}{\Rightarrow} \alpha X \beta, \text{con } \alpha, \beta \in (N' \cup \Sigma')^*$$

Es decir, N' y Σ' contendrán sólo aquellos símbolos que se puedan alcanzar desde el axioma.

El algoritmo es como sigue:

1. Inicializar $N' = \{S\}$, $P' = \emptyset$ y $\Sigma' = \emptyset$.
2. Para cada $A \in N'$, si $A \to w \in P$, entonces:
   - Añadir $A \to w$ a P'.
   - Para todo no terminal B en w, añadir B a N'.
   - Y para todo terminal σ en w, añadir σ a Σ'.
3. Repetir el paso anterior hasta que no se puedan añadir nuevas reglas a P'.

4 Simplificación de gramáticas independientes del contexto

Eliminación de símbolos no accesibles. Dada $G = (N, \Sigma, P, S)$ construimos una nueva GIC $G' = (N', \Sigma', P', S)$, tal que $L(G) = L(G')$, y donde:

$$\forall X \in (N' \cup \Sigma'), S \stackrel{*}{\Rightarrow} \alpha X \beta, \text{con } \alpha, \beta \in (N' \cup \Sigma')^*$$

Es decir, N' y Σ' contendrán sólo aquellos símbolos que se puedan alcanzar desde el axioma.

El algoritmo es como sigue:

1. Inicializar $N' = \{S\}$, $P' = \emptyset$ y $\Sigma' = \emptyset$.
2. Para cada $A \in N'$, si $A \to w \in P$, entonces:
   - Añadir $A \to w$ a P'.
   - Para todo no terminal B en w, añadir B a N'.
   - Y para todo terminal σ en w, añadir σ a Σ'.
3. Repetir el paso anterior hasta que no se puedan añadir nuevas reglas a P'.

Si aplicamos este algoritmo a la gramática del ejemplo anterior, obtenemos:

$$S \to Aa \mid B \qquad A \to aA \mid bA \mid B \qquad B \to b$$

donde se puede observar que hemos eliminado los símbolos C y d, y la producción $C \to abd$.

Esta garantizado que este algoritmo también termina, ya que N, Σ y P son finitos.

4 Simplificación de gramáticas independientes del contexto

Eliminación de ϵ-producciones. Las ϵ-producciones pueden resultar incómodas para ciertas tareas. Indudablemente, dada una GIC $G = (N, \Sigma, P, S)$, si $\epsilon \in L(G)$, al menos la producción $S \rightarrow \epsilon$ debe permanecer en P, pero el resto de ϵ-producciones pueden ser eliminadas.

4 Simplificación de gramáticas independientes del contexto

Eliminación de ϵ-producciones. Las ϵ-producciones pueden resultar incómodas para ciertas tareas. Indudablemente, dada una GIC $G = (N, \Sigma, P, S)$, si $\epsilon \in L(G)$, al menos la producción $S \to \epsilon$ debe permanecer en P, pero el resto de ϵ-producciones pueden ser eliminadas.

Para ello, definimos cuándo un símbolo es *anulable*. Dado $A \in N$, A es anulable si $A \stackrel{*}{\Rightarrow} \epsilon$. El siguiente algoritmo calcula en $\mathcal{N}$ el conjunto de todos los símbolos anulables de una gramática:

1. Inicializar $\mathcal{N}$ con todos los no terminales A para los cuales $A \to \epsilon \in P$.
2. Añadir los no terminales B tales que $B \to w \in P$ y todos los símbolos de w están en $\mathcal{N}$.
3. Repetir el paso anterior hasta que no se puedan añadir más no terminales a $\mathcal{N}$.

4 Simplificación de gramáticas independientes del contexto

Eliminación de ϵ-producciones. Las ϵ-producciones pueden resultar incómodas para ciertas tareas. Indudablemente, dada una GIC $G = (N, \Sigma, P, S)$, si $\epsilon \in L(G)$, al menos la producción $S \to \epsilon$ debe permanecer en P, pero el resto de ϵ-producciones pueden ser eliminadas.

Para ello, definimos cuándo un símbolo es *anulable*. Dado $A \in N$, A es anulable si $A \overset{*}{\Rightarrow} \epsilon$. El siguiente algoritmo calcula en $\mathcal{N}$ el conjunto de todos los símbolos anulables de una gramática:

1. Inicializar $\mathcal{N}$ con todos los no terminales A para los cuales $A \to \epsilon \in P$.
2. Añadir los no terminales B tales que $B \to w \in P$ y todos los símbolos de w están en $\mathcal{N}$.
3. Repetir el paso anterior hasta que no se puedan añadir más no terminales a $\mathcal{N}$.

Ahora, para eliminar las ϵ-producciones, consideramos primero aquellas gramáticas G tales que $\epsilon \notin L(G)$. Construimos $\mathcal{N}$ y después modificamos P como sigue. Sea $B \to X_1 X_2 \dots X_n \in P$. Entonces introducimos en P' todas las siguientes producciones:

$$B \to Y_1 Y_2 \dots Y_n \text{ tal que } Y_i = \begin{cases} - X_i, \text{ si } X_i \text{ no es anulable} \\ - X_i \text{ o } \epsilon, \text{ si } X_i \text{ es anulable} \\ - Y_i \text{ no puede ser } \epsilon \text{ para todo } i \end{cases}$$

Por ejemplo, si tenemos el siguiente conjunto de reglas:

$$B \to X_1 X_2 X_3 \qquad X_1 \to a \mid \epsilon \qquad X_2 \to aB \mid \epsilon \qquad X_3 \to aaB \mid \epsilon$$

este método las transforma en:

$$B \to X_1 X_2 X_3 \mid X_2 X_3 \mid X_1 X_3 \mid X_1 X_2 \mid X_3 \mid X_2 \mid X_1 \qquad X_1 \to a \qquad X_2 \to aB \mid a \qquad X_3 \to aaB \mid aa$$

Y finalmente, si $\epsilon \in L(G)$, simplemente añadimos la regla $S \to \epsilon$.

4 Simplificación de gramáticas independientes del contexto

Eliminación de reglas unitarias. No se trata de reglas inútiles, pero sí hacen la gramática innecesariamente compleja a veces. Para su eliminación, primero identificamos a qué símbolos se puede llegar desde uno dado usando sólo reglas unitarias:

$$unitario(A) = \{B \in N \mid A \stackrel{*}{\Rightarrow} B \text{ usando sólo reglas unitarias}\}$$

El $*$ significa 0 o más veces, lo que implica que $unitario(A)$ siempre contiene al propio A.

4 Simplificación de gramáticas independientes del contexto

Eliminación de reglas unitarias. No se trata de reglas inútiles, pero sí hacen la gramática innecesariamente compleja a veces. Para su eliminación, primero identificamos a qué símbolos se puede llegar desde uno dado usando sólo reglas unitarias:

$$unitario(A) = \{B \in N \mid A \stackrel{*}{\Rightarrow} B \text{ usando sólo reglas unitarias}\}$$

El $*$ significa 0 o más veces, lo que implica que $unitario(A)$ siempre contiene al propio A.

Después, calculamos P' como sigue:

1. Inicializar $P' = P$.
2. Para todo $A \in N$, obtener $unitario(A)$.
3. Para todo $A \in N$ tal que $unitario(A) \neq \{A\}$, para todo $B \in unitario(A)$, y para toda regla $B \to w \in P$, añadir $A \to w$ en P'.
4. Eliminar todas las producciones unitarias de P'.

4 Simplificación de gramáticas independientes del contexto

Eliminación de reglas unitarias. No se trata de reglas inútiles, pero sí hacen la gramática innecesariamente compleja a veces. Para su eliminación, primero identificamos a qué símbolos se puede llegar desde uno dado usando sólo reglas unitarias:

$$\mathit{unitario}(A) = \{B \in N \mid A \overset{*}{\Rightarrow} B \text{ usando sólo reglas unitarias}\}$$

El $*$ significa 0 o más veces, lo que implica que $\mathit{unitario}(A)$ siempre contiene al propio A.

Después, calculamos P' como sigue:

1. Inicializar $P' = P$.
2. Para todo $A \in N$, obtener $\mathit{unitario}(A)$.
3. Para todo $A \in N$ tal que $\mathit{unitario}(A) \neq \{A\}$, para todo $B \in \mathit{unitario}(A)$, y para toda regla $B \to w \in P$, añadir $A \to w$ en P'.
4. Eliminar todas las producciones unitarias de P'.

Por ejemplo, dada la gramática:

$$S \to A \mid Aa \qquad A \to B \qquad B \to C \mid b \qquad C \to D \mid ab \qquad D \to b$$

la aplicación de este algoritmo produce:

$$S \to Aa \mid b \mid ab \qquad A \to b \mid ab \qquad B \to b \mid ab \qquad C \to b \mid ab \qquad D \to b$$

Atención

Es importante tener en cuenta el orden de aplicación de todos estos métodos. Por ejemplo, ya en lo que se refiere a los dos primeros, cabe la posibilidad de que se produzcan resultados distintos si no se aplican en el orden descrito:

$$\left\{ \begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow a \end{array} \right\} \xrightarrow{1} \left\{ \begin{array}{l} S \rightarrow a \\ A \rightarrow a \end{array} \right\} \xrightarrow{2} \left\{ \; S \rightarrow a \; \right\}$$

$$\left\{ \begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow a \end{array} \right\} \xrightarrow{2} \left\{ \begin{array}{l} S \rightarrow AB \mid a \\ A \rightarrow a \end{array} \right\} \xrightarrow{1} \left\{ \begin{array}{l} S \rightarrow a \\ A \rightarrow a \end{array} \right\}$$

Por tanto, el **orden global de aplicación de los algoritmos de limpieza** debe ser el siguiente:

1. Eliminación de ϵ-producciones.
2. Eliminación de reglas unitarias.
3. Eliminación de símbolos inútiles.
4. Eliminación de símbolos no accesibles.

Definición

Dada una GIC G, siempre se puede escribir otra G' equivalente con producciones de la forma $A \to a$ o bien $A \to BC$. Se dice entonces que G' está en **forma normal de Chomsky** o **FNC**.

Las GIC,s en FNC son muy estructuradas y producen árboles de derivación siempre binarios. Además, el hecho de que cualquier GIC se pueda poner en FNC permitirá demostrar la existencia de un lema del bombeo para LIC,s y de los algoritmos de análisis sintáctico o *parsing*.

4 Simplificación de gramáticas independientes del contexto

Definición

Dada una GIC G, siempre se puede escribir otra G' equivalente con producciones de la forma $A \to a$ o bien $A \to BC$. Se dice entonces que G' está en **forma normal de Chomsky** o **FNC**.

Las GIC,s en FNC son muy estructuradas y producen árboles de derivación siempre binarios. Además, el hecho de que cualquier GIC se pueda poner en FNC permitirá demostrar la existencia de un lema del bombeo para LIC,s y de los algoritmos de análisis sintáctico o *parsing*.

El algoritmo de conversión a FNC es el siguiente. Suponemos G una gramática "limpia" (es decir, sin reglas-ϵ, ni unitarias, ni símbolos inútiles, ni no accesibles) y tal que $\epsilon \notin L(G)$. Por tanto, las reglas son de la forma $A \to w$, donde $|w| \geq 1$. Así pues:

- Si $|w| = 1$, necesariamente w es un terminal y la regla se deja tal cual.
- Si $|w| > 1$, entonces la regla es de la forma $A \to X_1 X_2 \dots X_n$:
  - Si X_i es un no terminal, no se toca.
  - Si X_i es el terminal σ, lo cambiamos por un nuevo no terminal C_σ y añadimos la regla $C_\sigma \to \sigma$.
  - Por tanto, siempre podemos hacer que $A \to B_1 B_2 \dots B_n$, donde todos los B_i son no terminales. Y ahora, simplemente "binarizamos" la regla:

$$A \to B_1 D_1 \qquad D_1 \to B_2 D_2 \qquad D_2 \to B_3 D_3 \qquad \dots \qquad D_{n-2} \to B_{n-1} B_n$$

Y si $\epsilon \in L(G)$, igual que antes, simplemente añadimos la regla $S \to \epsilon$.

A continuación, mostramos un ejemplo de aplicación de este algoritmo de conversión a FNC. Mediante dicho algoritmo, la siguiente gramática:

$$S \to bA \mid aB \qquad A \to bAA \mid aS \mid a \qquad B \to aBBB \mid bS \mid b$$

es transformada primero en:

$$S \to C_b A \mid C_a B \quad A \to C_b AA \mid C_a S \mid a \quad B \to C_a BBB \mid C_b S \mid b \quad C_a \to a \quad C_b \to b$$

y finalmente en:

$$S \to C_b A \mid C_a B$$

$$A \to C_b D_1 \mid C_a S \mid a \qquad D_1 \to AA$$

$$B \to C_a D_2 \mid C_b S \mid b \qquad D_2 \to BD_3 \qquad D_3 \to BB$$

$$C_a \to a$$

$$C_b \to b$$

Contenidos

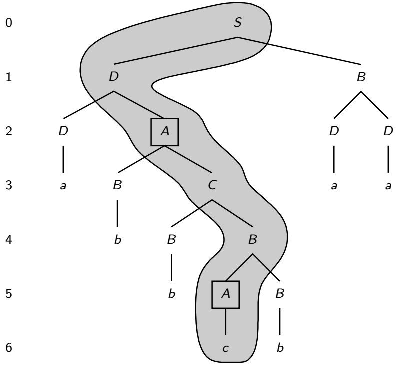
5 Propiedades de los lenguajes independientes del contexto

A partir de la FNC, introducimos ahora un razonamiento similar al lema del bombeo para los lenguajes regulares, que permitirá, en este caso, demostrar que algunos lenguajes no son LIC,s.

Sea una GIC donde $N = \{S, A, B, C, D\}$, es decir, $|N| = 5$. En el camino más largo de un árbol de profundidad 6, habrá 7 símbolos, de los cuales sólo el último es un terminal.

Así pues, hay al menos un símbolo no terminal que se repite en dicho camino. En este caso, consideremos, por ejemplo, el símbolo A.

La segunda aparición de A define entonces un punto en el que, en lugar de aplicar la reescritura mostrada en la figura para derivar la cadena $a\,bb\,c\,b\,aa$, podemos insertar de nuevo el primer subárbol encabezado por A para generar la cadena $a\,bb\,bb\,c\,b\,b\,aa$.



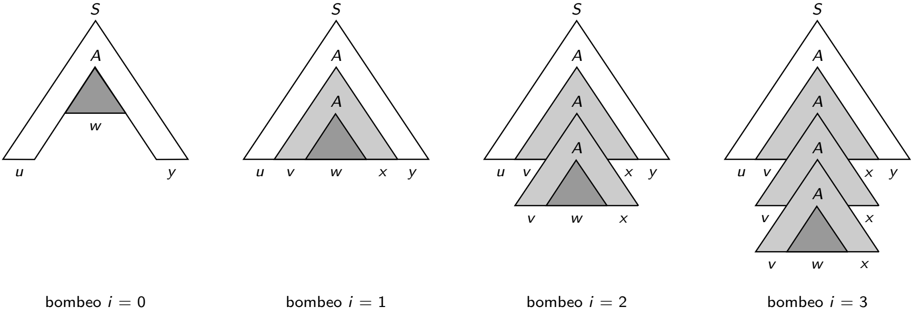
Y además, esta operación de inserción podría repetirse cualquier número de veces, haciendo posible la generación de cualquier cadena de la forma $a\,(bb)^i\,c\,(b)^i\,aa$, donde $i \geq 0$.

En general, este razonamiento se formaliza en el siguiente resultado.

Lema del Bombeo para LIC,s

Dado L un lenguaje independiente del contexto, existe un entero k para el cual, si $z \in L$, $|z| \geq k$, entonces la cadena z se puede descomponer en $z = uvwxy$, de tal forma que $|vwx| \leq k$ y $|v| + |x| > 0$, y cualquier cadena $uv^iwx^iy \in L$, $\forall i \geq 0$.

Gráficamente, para los bombeos $0 \leq i \leq 3$, podemos ilustrarlo así:



Lema del Bombeo para LIC,s (continuación)

Demostración:

Sea $G = (N, \Sigma, P, S)$ una GIC en FNC sin reglas ϵ y tal que $L(G) = L$ y $|N| = n$.
Si tomamos $z \in L$, $|z| \geq 2^n$, su árbol de derivación tendrá un camino con al menos $n+2$ nodos.
El último es un terminal, así que hay $n+1$ no terminales, y por tanto hay uno que se repite.
Si suponemos que A es ese símbolo no terminal, entonces tenemos que:

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = z \quad \text{(en el ejemplo, } u = a,\ v = bb,\ w = c,\ x = b,\ y = aa)$$

La repetición de A se resume en $A \overset{*}{\Rightarrow} vAx$. Aplicada i veces es $A \overset{*}{\Rightarrow} v^iAx^i$, $\forall i \geq 0$.
Y sustituyendo en la expresión anterior, vemos que $uv^iwx^iy \in L$, ya que $S \overset{*}{\Rightarrow} uv^iwx^iy$, $\forall i \geq 0$.

Lema del Bombeo para LIC,s (continuación)

Demostración:

Sea $G = (N, \Sigma, P, S)$ una GIC en FNC sin reglas ϵ y tal que $L(G) = L$ y $|N| = n$.
Si tomamos $z \in L$, $|z| \geq 2^n$, su árbol de derivación tendrá un camino con al menos $n+2$ nodos.
El último es un terminal, así que hay $n+1$ no terminales, y por tanto hay uno que se repite.
Si suponemos que A es ese símbolo no terminal, entonces tenemos que:

$$S \stackrel{*}{\Rightarrow} uAy \stackrel{*}{\Rightarrow} uvAxy \stackrel{*}{\Rightarrow} uvwxy = z \quad \text{(en el ejemplo, } u = a,\ v = bb,\ w = c,\ x = b,\ y = aa\text{)}$$

La repetición de A se resume en $A \stackrel{*}{\Rightarrow} vAx$. Aplicada i veces es $A \stackrel{*}{\Rightarrow} v^iAx^i$, $\forall i \geq 0$.
Y sustituyendo en la expresión anterior, vemos que $uv^iwx^iy \in L$, ya que $S \stackrel{*}{\Rightarrow} uv^iwx^iy$, $\forall i \geq 0$.

Verifiquemos ahora las condiciones:

- $|vwx| \leq k = 2^n$
  Estudiando el camino más largo hacia arriba, cuando se repite un no terminal, ese subcamino tendrá como mucho $n+1$ nodos. Por tanto, el subárbol que se replica al bombear tendrá como mucho 2^n hojas de símbolos terminales.
- $|v| + |x| > 0$
  En $A \stackrel{*}{\Rightarrow} vAx$, necesariamente $A \Rightarrow BC \stackrel{*}{\Rightarrow} vAx$.
  Así que o bien $B \stackrel{*}{\Rightarrow} v$ y $C \stackrel{*}{\Rightarrow} Ax$, o bien $B \stackrel{*}{\Rightarrow} vA$ y $C \stackrel{*}{\Rightarrow} x$.
  Si $B = A$ o si $C = A$, v o x pueden ser ϵ, pero no ambas, ya que G no tiene reglas ϵ. $\square$

Ejercicio

Demuestre mediante el lema del bombeo que $L = \{a^i b^i c^i \mid i \geq 0\}$ no es un LIC.

Ejercicio

Demuestre mediante el lema del bombeo que $L = \{a^i b^i c^i \mid i \geq 0\}$ no es un LIC.

<u>Solución</u>:

Suponemos que L es un LIC y sea k su constante asociada según el lema del bombeo.
Tomemos entonces una cadena $z \in L$ tal que $|z| \geq k$, por ejemplo, $z = a^k b^k c^k$.
Las posibles descomposiciones de z en $uvwxy$, tales que $|vwx| \leq k$ y $|v| + |x| > 0$, son:

- La porción vwx está compuesta sólo por $a$es. En este caso, cualquier bombeo $uv^i wx^i y$, $i > 1$, produce cadenas con más $a$es que $b$es y $c$es, y que por tanto no pertenecen a L.
- La porción vwx está compuesta sólo por $b$es. Este caso es similar al anterior. Lo que antes ocurría con las $a$es ocurre ahora con las $b$es.
- La porción vwx está compuesta sólo por $c$es. Este caso también es similar al anterior. Lo que antes ocurría con las $b$es ocurre ahora con las $c$es.
- La porción vwx está compuesta por $a$es y por $b$es. Detectamos entonces tres subcasos:
    - La subcadena v está formada sólo por $a$es, y la subcadena x sólo por $b$es. Al bombear, aumentamos las $a$es o las $b$es o ambas, pero nunca las $c$es.
    - La subcadena v está formada por $a$es y por $b$es, y la subcadena x sólo por $b$es. Cualquier bombeo hará que las $a$es y las $b$es se mezclen.
    - La subcadena v está formada sólo por $a$es, y la subcadena x por $a$es y por $b$es. Igual que antes, cualquier bombeo mezcla las $a$es y las $b$es.

.../...

Ejercicio (continuación)

- La porción vwx está compuesta por $b$es y por $c$es. Este caso es similar al anterior. Lo que antes ocurría con las $a$es y las $b$es ocurre ahora con las $b$es y las $c$es.

La porción vwx no puede contener $a$es, $b$es y $c$es, ya que entonces tendría k $b$es y algún símbolo más, es decir, tendría longitud al menos $k+2$, y no verificaría la condición $|vwx| \leq k$.

Así pues, hemos analizado todos los casos de descomposición de z y, en todos ellos, hemos encontrado al menos un bombeo que produce cadenas fuera de L, con lo cual L no es un LIC.

Ejercicio (continuación)

- La porción vwx está compuesta por $b$es y por $c$es. Este caso es similar al anterior. Lo que antes ocurría con las $a$es y las $b$es ocurre ahora con las $b$es y las $c$es.

La porción vwx no puede contener $a$es, $b$es y $c$es, ya que entonces tendría k $b$es y algún símbolo más, es decir, tendría longitud al menos $k + 2$, y no verificaría la condición $|vwx| \leq k$.

Así pues, hemos analizado todos los casos de descomposición de z y, en todos ellos, hemos encontrado al menos un bombeo que produce cadenas fuera de L, con lo cual L no es un LIC.

Ejercicio

Demuestre mediante el lema del bombeo que $L = \{a^p \mid p \text{ es un número primo}\}$ no es un LIC.

Ejercicio (continuación)

- La porción vwx está compuesta por $b$es y por $c$es. Este caso es similar al anterior. Lo que antes ocurría con las $a$es y las $b$es ocurre ahora con las $b$es y las $c$es.

La porción vwx no puede contener $a$es, $b$es y $c$es, ya que entonces tendría k $b$es y algún símbolo más, es decir, tendría longitud al menos $k+2$, y no verificaría la condición $|vwx| \leq k$.

Así pues, hemos analizado todos los casos de descomposición de z y, en todos ellos, hemos encontrado al menos un bombeo que produce cadenas fuera de L, con lo cual L no es un LIC.

Ejercicio

Demuestre mediante el lema del bombeo que $L = \{a^p \mid p \text{ es un número primo}\}$ no es un LIC.

Solución:

Suponemos que L es un LIC y sea k su constante asociada según el lema del bombeo.
Tomemos $z = a^n$ donde n es un número primo mayor que k.
En cualquier bombeo uv^iwx^iy, las subcadenas u, w e y siempre permanecen constantes.
Llamemos m a la longitud $|u| + |w| + |y|$, y por tanto $n - m$ a la longitud $|v| + |x|$.
La longitud de cualquier bombeo i será entonces de la forma $|uv^iwx^iy| = m + i\,(n-m)$.
Si bombeamos exactamente $i = m$ veces, obtenemos una cadena de longitud $m\,(1 + n - m)$.
Esta longitud no es un número primo, ya que $n - m \neq 0$.
Así pues, el bombeo $i = m$ produce una cadena fuera de L, con lo cual L no es un LIC.

Teorema

Dada una gramática $G = (N, \Sigma, P, S)$, donde $|N| = n$, $L(G)$ es infinito si y sólo si existe una cadena $z \in L(G)$ tal que $2^{n-1} < |z| \leq 2^{n-1} + 2^n$. $\square$

5 Propiedades de los lenguajes independientes del contexto

Teorema

Dada una gramática $G = (N, \Sigma, P, S)$, donde $|N| = n$, $L(G)$ es infinito si y sólo si existe una cadena $z \in L(G)$ tal que $2^{n-1} < |z| \leq 2^{n-1} + 2^n$. $\square$

Teorema

Los LIC,s son cerrados para la operación de unión.

Teorema

Dada una gramática $G = (N, \Sigma, P, S)$, donde $|N| = n$, $L(G)$ es infinito si y sólo si existe una cadena $z \in L(G)$ tal que $2^{n-1} < |z| \leq 2^{n-1} + 2^n$. $\square$

Teorema

Los LIC,s son cerrados para la operación de unión.

Demostración:

Sean L_1 y L_2 dos LIC,s generados por las GIC,s $G_1 = (N_1, \Sigma_1, P_1, S_1)$ y $G_2 = (N_2, \Sigma_2, P_2, S_2)$, respectivamente. Podemos construir $G = (N, \Sigma, P, S)$, tal que $L(G) = L_1 \cup L_2$, como sigue:

$$N = N_1 \cup N_2 \cup \{S\} \qquad \Sigma = \Sigma_1 \cup \Sigma_2 \qquad P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\} \qquad S = S$$

$\square$

Teorema

Dada una gramática $G = (N, \Sigma, P, S)$, donde $|N| = n$, $L(G)$ es infinito si y sólo si existe una cadena $z \in L(G)$ tal que $2^{n-1} < |z| \leq 2^{n-1} + 2^n$. $\square$

Teorema

Los LIC,s son cerrados para la operación de unión.

Demostración:

Sean L_1 y L_2 dos LIC,s generados por las GIC,s $G_1 = (N_1, \Sigma_1, P_1, S_1)$ y $G_2 = (N_2, \Sigma_2, P_2, S_2)$, respectivamente. Podemos construir $G = (N, \Sigma, P, S)$, tal que $L(G) = L_1 \cup L_2$, como sigue:

$$N = N_1 \cup N_2 \cup \{S\} \qquad \Sigma = \Sigma_1 \cup \Sigma_2 \qquad P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\} \qquad S = S$$

$\square$

Teorema

Los LIC,s son cerrados para la operación de concatenación.

Teorema

Dada una gramática $G = (N, \Sigma, P, S)$, donde $|N| = n$, $L(G)$ es infinito si y sólo si existe una cadena $z \in L(G)$ tal que $2^{n-1} < |z| \leq 2^{n-1} + 2^n$. □

Teorema

Los LIC,s son cerrados para la operación de unión.

Demostración:

Sean L_1 y L_2 dos LIC,s generados por las GIC,s $G_1 = (N_1, \Sigma_1, P_1, S_1)$ y $G_2 = (N_2, \Sigma_2, P_2, S_2)$, respectivamente. Podemos construir $G = (N, \Sigma, P, S)$, tal que $L(G) = L_1 \cup L_2$, como sigue:

$$N = N_1 \cup N_2 \cup \{S\} \qquad \Sigma = \Sigma_1 \cup \Sigma_2 \qquad P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\} \qquad S = S$$

□

Teorema

Los LIC,s son cerrados para la operación de concatenación.

Demostración:

Procedemos de manera similar al resultado anterior, pero la nueva regla de producción a añadir en este caso es $S \to S_1 S_2$. □

Teorema

Los LIC,s son cerrados para la operación de cierre.

Teorema

Los LIC,s son cerrados para la operación de cierre.

Demostración:

Sea L un LIC generado por la GIC $G = (N, \Sigma, P, S)$. Podemos construir $G' = (N', \Sigma, P', S')$, tal que $L(G') = L^*$, como sigue:

$$N' = N \cup \{S', T\} \qquad P = P \cup \{S' \to \epsilon \mid ST,\ T \to \epsilon \mid ST\}$$

□

Teorema

Los LIC,s son cerrados para la operación de cierre.

Demostración:

Sea L un LIC generado por la GIC $G = (N, \Sigma, P, S)$. Podemos construir $G' = (N', \Sigma, P', S')$, tal que $L(G') = L^*$, como sigue:

$$N' = N \cup \{S', T\} \qquad P = P \cup \{S' \to \epsilon \mid ST,\ T \to \epsilon \mid ST\}$$

□

Ejercicio

Demuestre que los LIC,s no son cerrados para la operación de intersección.

Teorema

Los LIC,s son cerrados para la operación de cierre.

Demostración:

Sea L un LIC generado por la GIC $G = (N, \Sigma, P, S)$. Podemos construir $G' = (N', \Sigma, P', S')$, tal que $L(G') = L^*$, como sigue:

$$N' = N \cup \{S', T\} \qquad P = P \cup \{S' \to \epsilon \mid ST,\ T \to \epsilon \mid ST\}$$

□

Ejercicio

Demuestre que los LIC,s no son cerrados para la operación de intersección.

Solución:

Para verlo, basta con proporcionar un contraejemplo.
$L_1 = \{a^i b^j c^j \mid i, j \geq 0\}$ es un LIC, ya que puede ser generado por la GIC:

$$S \to AC \qquad A \to aA \mid \epsilon \qquad C \to bCc \mid \epsilon$$

Lo mismo ocurre con $L_2 = \{a^i b^i c^j \mid i, j \geq 0\}$, que puede ser generado por:

$$S \to AC \qquad A \to aAb \mid \epsilon \qquad C \to cC \mid \epsilon$$

Sin embargo, $L_1 \cap L_2 = \{a^k b^k c^k \mid k \geq 0\}$, que ya demostramos que no es un LIC.

A partir del resultado anterior, se deduce que el complementario de un LIC, en general, tampoco es un LIC, ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

A partir del resultado anterior, se deduce que el complementario de un LIC, en general, tampoco es un LIC, ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

No obstante, si intersecamos un LIC con un lenguaje regular, el resultado sí es un LIC, tal y como veremos más adelante.

5 Propiedades de los lenguajes independientes del contexto

A partir del resultado anterior, se deduce que el complementario de un LIC, en general, tampoco es un LIC, ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

No obstante, si intersecamos un LIC con un lenguaje regular, el resultado sí es un LIC, tal y como veremos más adelante.

Teorema

Sea $G = (N, \Sigma, P, S)$ una GIC en FNC, y w una cadena de Σ^*. Para todo símbolo no terminal $A \in N$ es posible saber si se verifica que $A \overset{*}{\Rightarrow} x$, donde x es cualquier subcadena de w.

5 Propiedades de los lenguajes independientes del contexto

A partir del resultado anterior, se deduce que el complementario de un LIC, en general, tampoco es un LIC, ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

No obstante, si intersecamos un LIC con un lenguaje regular, el resultado sí es un LIC, tal y como veremos más adelante.

Teorema

Sea $G = (N, \Sigma, P, S)$ una GIC en FNC, y w una cadena de Σ^*. Para todo símbolo no terminal $A \in N$ es posible saber si se verifica que $A \stackrel{*}{\Rightarrow} x$, donde x es cualquier subcadena de w.

Demostración:

Supongamos $|x| = n$ y $x = w_{1,n}$, es decir, con $w_{i,j}$ denotamos la subcadena que empieza en i y tiene longitud j. La prueba se hace por inducción en la longitud j.

- Si $j = 1$, $w_{i,j}$ es un terminal, y comprobar si $A \stackrel{*}{\Rightarrow} w_{i,j}$ es comprobar si $A \rightarrow w_{i,j} \in P$.
- Lo suponemos cierto para cualquier longitud $j > 1$.
- Para $j+1$, $A \stackrel{*}{\Rightarrow} w_{i,j+1}$ es cierto si $A \rightarrow BC$, $B \stackrel{*}{\Rightarrow} w_{i,k}$ y $C \stackrel{*}{\Rightarrow} w_{i+k,j-k+1}$. Por hipótesis de inducción, estas derivaciones también son verificables ya que tienen longitud $< j+1$. $\square$

5 Propiedades de los lenguajes independientes del contexto

A partir del resultado anterior, se deduce que el complementario de un LIC, en general, tampoco es un LIC, ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

No obstante, si intersecamos un LIC con un lenguaje regular, el resultado sí es un LIC, tal y como veremos más adelante.

Teorema

Sea $G = (N, \Sigma, P, S)$ una GIC en FNC, y w una cadena de Σ^*. Para todo símbolo no terminal $A \in N$ es posible saber si se verifica que $A \overset{*}{\Rightarrow} x$, donde x es cualquier subcadena de w.

Demostración:

Supongamos $|x| = n$ y $x = w_{1,n}$, es decir, con $w_{i,j}$ denotamos la subcadena que empieza en i y tiene longitud j. La prueba se hace por inducción en la longitud j.

- Si $j = 1$, $w_{i,j}$ es un terminal, y comprobar si $A \overset{*}{\Rightarrow} w_{i,j}$ es comprobar si $A \to w_{i,j} \in P$.
- Lo suponemos cierto para cualquier longitud $j > 1$.
- Para $j + 1$, $A \overset{*}{\Rightarrow} w_{i,j+1}$ es cierto si $A \to BC$, $B \overset{*}{\Rightarrow} w_{i,k}$ y $C \overset{*}{\Rightarrow} w_{i+k,j-k+1}$. Por hipótesis de inducción, estas derivaciones también son verificables ya que tienen longitud $< j + 1$. $\square$

En particular, podemos saber entonces si $S \overset{*}{\Rightarrow} w$. Es decir, este resultado indica que existen métodos para saber si cualquier cadena $w \in \Sigma^*$ pertenece o no a $L(G)$. Precisamente, la siguiente sección considera varios algoritmos que resuelven esta cuestión.

Contenidos

6 Algoritmos de análisis sintáctico

Dada una GIC y una cadena de símbolos, queremos encontrar las estructuras sintácticas de dicha cadena de acuerdo con la gramática. Los algoritmos que realizan esta tarea se denominan **analizadores sintácticos** o **parsers**, y pueden clasificarse en dos grandes grupos:

- Los analizadores **descendentes** son aquéllos que parten del axioma de la gramática y exploran los posibles árboles de derivación, intentando obtener una secuencia de reglas que genere la cadena de símbolos a procesar.
- Los analizadores **ascendentes** utilizan la misma estrategia, pero en sentido contrario. Es decir, parten de la cadena de símbolos e intentan obtener el axioma de la gramática.

6 Algoritmos de análisis sintáctico

Dada una GIC y una cadena de símbolos, queremos encontrar las estructuras sintácticas de dicha cadena de acuerdo con la gramática. Los algoritmos que realizan esta tarea se denominan **analizadores sintácticos** o **parsers**, y pueden clasificarse en dos grandes grupos:

- Los analizadores **descendentes** son aquéllos que parten del axioma de la gramática y exploran los posibles árboles de derivación, intentando obtener una secuencia de reglas que genere la cadena de símbolos a procesar.
- Los analizadores **ascendentes** utilizan la misma estrategia, pero en sentido contrario. Es decir, parten de la cadena de símbolos e intentan obtener el axioma de la gramática.

Respecto a los **analizadores descendentes**, las dos principales aproximaciones son:

- Utilización de una estrutura de cola para recorrer **en anchura** el espacio de posibles derivaciones. Si existe una derivación para la cadena de entrada, el algoritmo la encontrará. No obstante, dicho espacio de derivaciones podría crecer demasiado, en cuyo caso, el algoritmo consumirá una gran cantidad de recursos.
- Utilización de una estrutura de pila para recorrer **en profundidad** el espacio de derivaciones. Este tipo de algoritmos consume muy pocos recursos. Sin embargo, dependiendo del orden de aplicación de las reglas, la recursividad podría provocar que el algoritmo se quede procesando indefinidamente una cierta rama del árbol de derivaciones sin llegar a ninguna solución, aun cuando la solución está presente en otra rama del árbol.

Respecto a los **analizadores ascendentes**, uno de los mecanismos más sencillos e intuitivos es el tradicionalmente conocido como **algoritmo CYK** o *algoritmo Cocke-Younger-Kasami*, aunque fue descubierto de manera independiente por distintas personas.

6 Algoritmos de análisis sintáctico

Respecto a los **analizadores ascendentes**, uno de los mecanismos más sencillos e intuitivos es el tradicionalmente conocido como **algoritmo CYK** o *algoritmo Cocke-Younger-Kasami*, aunque fue descubierto de manera independiente por distintas personas.

La esencia del algoritmo CYK consiste en:

- Se construye una tabla de análisis triangular, cuyas celdas se denotan por N_{ij}, para $1 \leq i \leq n - j + 1$ y $1 \leq j \leq n$, donde n es el número de símbolos de la cadena a analizar.
- Cada celda contendrá un subconjunto de los símbolos no terminales de la gramática.
- Un símbolo no terminal A estará en N_{ij} si y sólo si $A \stackrel{*}{\Rightarrow} w_i, w_{i+1}, \ldots, w_{i+j-1}$, es decir, si A deriva en un número finito de pasos la subcadena que comienza en la posición i y contiene j símbolos.
- La cadena pertenece al lenguaje generado por la gramática si al final del proceso el axioma se encuentra en la celda N_{1n}.

6 Algoritmos de análisis sintáctico

Considerando una cadena de entrada $w = w_1, w_2, \ldots, w_n$ y una gramática en forma normal de Chomsky, sin ϵ-producciones, $G = (N, T, P, S)$, la **descripción formal del algoritmo CYK** es la siguiente:

1. Se inicializa la primera fila de la tabla de análisis, utilizando las reglas que generan directamente los símbolos terminales, como sigue:

$$N_{i1} = \{A \mid A \to w_{i1} \in P\},\ 1 \leq i \leq n$$

2. Para $j = 2, 3, \ldots, n$, hacer lo siguiente:
   - Para $i = 1, 2, \ldots, n - j + 1$, hacer lo siguiente:
     - Inicializar N_{ij} al conjunto vacío.
     - Para $k = 1, 2, \ldots, j - 1$, añadir a N_{ij} todos los símbolos no terminales A para los cuales $A \to BC \in P$, con $B \in N_{ik}$ y $C \in N_{(i+k)(j-k)}$.

3. La cadena w pertenece a $L(G)$ si y sólo si $S \in N_{1n}$.

La siguiente figura muestra la tabla de análisis CYK para la gramática

$$S \to AB \mid BC \qquad A \to BA \mid a \qquad B \to CC \mid b \qquad C \to AB \mid a$$

y la cadena $w = bbab$. Dado que $S \in N_{14}$, w pertenece al lenguaje generado por esta GIC.

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

6 Algoritmos de análisis sintáctico

Aunque resulta de gran interés por su simplicidad, el algoritmo CYK tiene **inconvenientes**:

- La versión original sólo trabaja con **GIC,s en FNC**.
- La **complejidad temporal es** $\mathcal{O}(\mathbf{n^3})$, donde *n* es el número de símbolos de la cadena a analizar, independientemente de dicha cadena.
- La **complejidad espacial es** $\mathcal{O}(\mathbf{n^2})$, independientemente también de la cadena a analizar.

6 Algoritmos de análisis sintáctico

Aunque resulta de gran interés por su simplicidad, el algoritmo CYK tiene **inconvenientes**:

- La versión original sólo trabaja con **GIC,s en FNC**.
- La **complejidad temporal es** $\mathcal{O}(\mathbf{n^3})$, donde n es el número de símbolos de la cadena a analizar, independientemente de dicha cadena.
- La **complejidad espacial es** $\mathcal{O}(\mathbf{n^2})$, independientemente también de la cadena a analizar.

Aún con todo esto, se pueden señalar también las siguientes **ventajas**:

- El algoritmo CYK se puede extender para su funcionamiento sobre **GIC,s arbitrarias**.
- Presenta un esquema natural de **paralelización** que permite rebajar sus complejidades.
- La presencia de un símbolo no terminal en una celda N_{ij} implica la existencia de un subárbol encabezado por dicho símbolo, que cubre la subcadena $w_i, w_{i+1}, \ldots, w_{i+j-1}$. Así pues, la **extracción de los árboles sintácticos** constituye una tarea realmente sencilla, no sólo en el caso de los árboles de cobertura total correspondientes a la celda superior, sino también para cualquier otro subárbol de cualquier otra celda intermedia.
- Si unimos a lo anterior la posibilidad de trabajar con **GIC,s probabilísticas**, tendremos un marco especialmente adaptado para el procesamiento no sólo de lenguajes formales, sino también de muchos de los aspectos sintácticos que presentan las lenguas naturales.

6 Algoritmos de análisis sintáctico

Aunque resulta de gran interés por su simplicidad, el algoritmo CYK tiene **inconvenientes**:

- La versión original sólo trabaja con **GIC,s en FNC**.
- La **complejidad temporal es $\mathcal{O}(\mathbf{n^3})$**, donde n es el número de símbolos de la cadena a analizar, independientemente de dicha cadena.
- La **complejidad espacial es $\mathcal{O}(\mathbf{n^2})$**, independientemente también de la cadena a analizar.

Aún con todo esto, se pueden señalar también las siguientes **ventajas**:

- El algoritmo CYK se puede extender para su funcionamiento sobre **GIC,s arbitrarias**.
- Presenta un esquema natural de **paralelización** que permite rebajar sus complejidades.
- La presencia de un símbolo no terminal en una celda N_{ij} implica la existencia de un subárbol encabezado por dicho símbolo, que cubre la subcadena $w_i, w_{i+1}, \ldots, w_{i+j-1}$. Así pues, la **extracción de los árboles sintácticos** constituye una tarea realmente sencilla, no sólo en el caso de los árboles de cobertura total correspondientes a la celda superior, sino también para cualquier otro subárbol de cualquier otra celda intermedia.
- Si unimos a lo anterior la posibilidad de trabajar con **GIC,s probabilísticas**, tendremos un marco especialmente adaptado para el procesamiento no sólo de lenguajes formales, sino también de muchos de los aspectos sintácticos que presentan las lenguas naturales.

En cualquier caso, la tarea del análisis sintáctico ha constituido un área de investigación de gran actividad, por lo que existen muchos otros algoritmos de *parsing*, tanto ascendentes como descendentes, cuyo estudio pertenece al ámbito de otras asignaturas. Aquí nos hemos limitado a indicar los fundamentos y aplicaciones de los más importantes.

Contenidos

Definición

Un **autómata de pila no determinista** o **APN** M es una colección de siete elementos, $M = (Q, \Sigma, \Gamma, s, Z, F, \Delta)$, donde:

- Q es un conjunto finito de estados.
- Σ es el alfabeto de los símbolos de entrada.
- Γ es el alfabeto de los símbolos de la pila.
- $s \in Q$ es el estado inicial del autómata.
- $Z \in \Gamma$ es el símbolo inicial de la pila (lo suponemos siempre presente, aunque esto no es imprescindible: se podría añadir un estado extra con una transición que lo primero que haría sería insertar Z en la pila).
- $F \subseteq Q$ es el conjunto de estados finales o de aceptación.
- Δ se define mediante ternas del tipo $(q, \sigma, \gamma) \to (p, w)$, donde $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $\gamma \in \Gamma \cup \{\epsilon\}$, $p \in Q$ y $w \in \Gamma^*$. Es decir, desde el estado q, con el símbolo de entrada σ y con γ en la cima de la pila, el autómata pasa al estado p y cambia γ por w. Por tanto, $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to 2^{Q \times \Gamma^*}$.

7 Autómatas de pila

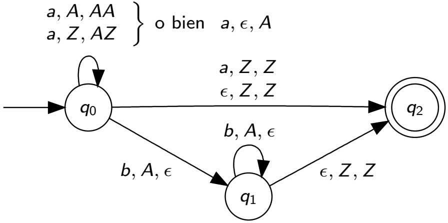
Definición

Un **autómata de pila no determinista** o **APN** M es una colección de siete elementos, $M = (Q, \Sigma, \Gamma, s, Z, F, \Delta)$, donde:

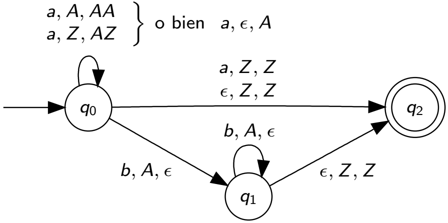
- Q es un conjunto finito de estados.
- Σ es el alfabeto de los símbolos de entrada.
- Γ es el alfabeto de los símbolos de la pila.
- $s \in Q$ es el estado inicial del autómata.
- $Z \in \Gamma$ es el símbolo inicial de la pila (lo suponemos siempre presente, aunque esto no es imprescindible: se podría añadir un estado extra con una transición que lo primero que haría sería insertar Z en la pila).
- $F \subseteq Q$ es el conjunto de estados finales o de aceptación.
- Δ se define mediante ternas del tipo $(q, \sigma, \gamma) \to (p, w)$, donde $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $\gamma \in \Gamma \cup \{\epsilon\}$, $p \in Q$ y $w \in \Gamma^*$. Es decir, desde el estado q, con el símbolo de entrada σ y con γ en la cima de la pila, el autómata pasa al estado p y cambia γ por w. Por tanto, $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to 2^{Q \times \Gamma^*}$.

Al igual que los autómatas finitos, los AP,s pueden representarse también de forma tabular o mediante un grafo.

Por ejemplo, la siguiente figura muestra un APN que acepta $\{a^n b^n \mid n \geq 0\} \cup \{a\}$:



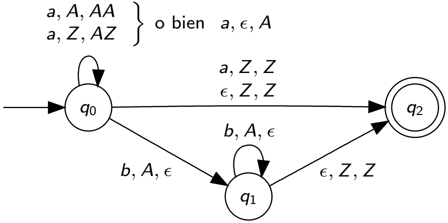
Por ejemplo, la siguiente figura muestra un APN que acepta $\{a^n b^n \mid n \geq 0\} \cup \{a\}$:



Las **configuraciones instantáneas** serán ahora ternas de la forma (q, w, u), donde q es el estado actual, w es la cadena de símbolos que queda por procesar, y u es el contenido de la pila. Según esto, el procesamiento de la cadena *aabb* podría describirse como sigue:

$$
\begin{array}{llll}
\{(q_0, aabb, Z)\} & \vdash & \{(q_0, abb, AZ,), (q_2, abb, Z)\} & \vdash & \{(q_0, bb, AAZ)\} \\
& \vdash & \{(q_1, b, AZ)\} & \vdash & \{(q_1, \epsilon, Z)\} \\
& \vdash & \{(q_2, \epsilon, Z)\} & &
\end{array}
$$

Por ejemplo, la siguiente figura muestra un APN que acepta $\{a^n b^n \mid n \geq 0\} \cup \{a\}$:



Las **configuraciones instantáneas** serán ahora ternas de la forma (q, w, u), donde q es el estado actual, w es la cadena de símbolos que queda por procesar, y u es el contenido de la pila. Según esto, el procesamiento de la cadena *aabb* podría describirse como sigue:

$$
\begin{array}{lllll}
\{(q_0, aabb, Z)\} & \vdash & \{(q_0, abb, AZ,), (q_2, abb, Z)\} & \vdash & \{(q_0, bb, AAZ)\} \\
 & \vdash & \{(q_1, b, AZ)\} & \vdash & \{(q_1, \epsilon, Z)\} \\
 & \vdash & \{(q_2, \epsilon, Z)\} & &
\end{array}
$$

Definición

El **lenguaje aceptado por un autómata de pila** está formado por las cadenas que al procesarse totalmente lo hacen llegar a un estado final, pudiendo la pila quedar vacía o no.

La relación que existe entre los AP,s y los LIC,s se establece mediante el siguiente resultado.

Teorema

Dada una GIC $G = (N, \Sigma, P, S)$, siempre se puede construir un APN M tal que $L(M) = L(G)$. $\square$

7 Autómatas de pila

La relación que existe entre los AP,s y los LIC,s se establece mediante el siguiente resultado.

Teorema

Dada una GIC $G = (N, \Sigma, P, S)$, siempre se puede construir un APN M tal que $L(M) = L(G)$. $\square$

La idea de esta construcción consiste en trabajar sobre cada producción con ayuda de la pila. Los elementos del AP serán $Q = \{q_1, q_2, q_3\}$, $\Sigma = \Sigma$, $\Gamma = N \cup \Sigma \cup \{Z\}$, $F = \{q_3\}$, $s = q_1$, $Z = Z$, y Δ se construye de manera que el AP realice las siguientes tareas:

- Introducir en la pila el axioma de la gramática.
- Cambiar no terminales por reglas: si en la cima de la pila tenemos un no terminal, buscamos qué reglas lo reescriben y las aplicamos.
- Comparar terminales: si en la cima de la pila tenemos un terminal, se compara con el primer símbolo de la cadena de entrada que quede por procesar, y si coinciden se eliminan de la pila y de la entrada, respectivamente.

7 Autómatas de pila

La relación que existe entre los AP,s y los LIC,s se establece mediante el siguiente resultado.

Teorema

Dada una GIC $G = (N, \Sigma, P, S)$, siempre se puede construir un APN M tal que $L(M) = L(G)$. $\square$

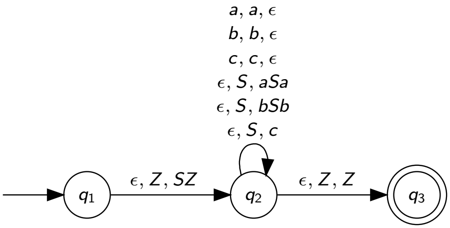
La idea de esta construcción consiste en trabajar sobre cada producción con ayuda de la pila. Los elementos del AP serán $Q = \{q_1, q_2, q_3\}$, $\Sigma = \Sigma$, $\Gamma = N \cup \Sigma \cup \{Z\}$, $F = \{q_3\}$, $s = q_1$, $Z = Z$, y Δ se construye de manera que el AP realice las siguientes tareas:

- Introducir en la pila el axioma de la gramática.
- Cambiar no terminales por reglas: si en la cima de la pila tenemos un no terminal, buscamos qué reglas lo reescriben y las aplicamos.
- Comparar terminales: si en la cima de la pila tenemos un terminal, se compara con el primer símbolo de la cadena de entrada que quede por procesar, y si coinciden se eliminan de la pila y de la entrada, respectivamente.

Habrá entonces cuatro tipos de transiciones:

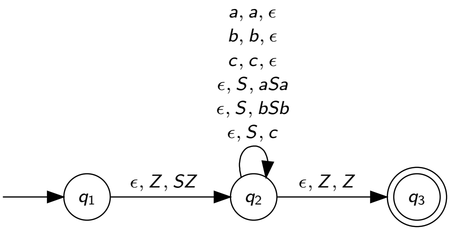
1. $\Delta(q_1, \epsilon, Z) = \{(q_2, SZ)\}$.
2. $\Delta(q_2, \epsilon, A) = \{(q_2, w) \mid A \to w \in P\}$, $\forall A \in N$.
3. $\Delta(q_2, a, a) = \{(q_2, \epsilon)\}$, $\forall a \in \Sigma$.
4. $\Delta(q_2, \epsilon, Z) = \{(q_3, Z)\}$.

A continuación mostramos el APN que se obtiene a partir de la GIC $S \to aSa \mid bSb \mid c$ y la secuencia de configuraciones instantáneas del reconocimiento de la cadena *abcba*, la cual se deriva de la forma $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abcba$.



$$
\begin{array}{ll}
 & (q_1, abcba, Z) \\
\vdash & (q_2, abcba, SZ) \\
\vdash & (q_2, abcba, aSaZ) \\
\vdash & (q_2, bcba, SaZ) \\
\vdash & (q_2, bcba, bSbaZ) \\
\vdash & (q_2, cba, SbaZ) \\
\vdash & (q_2, cba, cbaZ) \\
\vdash & (q_2, ba, baZ) \\
\vdash & (q_2, a, aZ) \\
\vdash & (q_2, \epsilon, Z) \\
\vdash & (q_3, \epsilon, Z)
\end{array}
$$

A continuación mostramos el APN que se obtiene a partir de la GIC $S \to aSa \mid bSb \mid c$ y la secuencia de configuraciones instantáneas del reconocimiento de la cadena *abcba*, la cual se deriva de la forma $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abcba$.



$$
\begin{array}{ll}
 & (q_1, abcba, Z) \\
\vdash & (q_2, abcba, SZ) \\
\vdash & (q_2, abcba, aSaZ) \\
\vdash & (q_2, bcba, SaZ) \\
\vdash & (q_2, bcba, bSbaZ) \\
\vdash & (q_2, cba, SbaZ) \\
\vdash & (q_2, cba, cbaZ) \\
\vdash & (q_2, ba, baZ) \\
\vdash & (q_2, a, aZ) \\
\vdash & (q_2, \epsilon, Z) \\
\vdash & (q_3, \epsilon, Z)
\end{array}
$$

Es una traza guiada que incluye sólo las configuraciones que llevan a la aceptación, y muestra que el orden de uso de las reglas en la pila es el mismo que en la derivación de la cadena.

Si pudiéramos entonces recuperar las operaciones de la pila, obtendríamos la derivación completa. Esto podría hacerse mediante técnicas de programación dinámica.

Y, en lugar de hacer *backtracking* o de explorar las configuraciones en anchura, sería cómodo también mirar la cadena hacia adelante utilizando lo que se denomina un *lock-ahead* para seleccionar las reglas. Más adelante veremos que la forma normal de Greibach facilita esta tarea.

7 Autómatas de pila

Sabiendo que a partir de una GIC se puede obtener un AP equivalente, ahora la pregunta es:

- El lenguaje que acepta un AP, ¿es siempre un LIC?
- La respuesta es que sí, ya que dado un AP, se puede también obtener una GIC equivalente, cuyas reglas se calculan a partir de las transiciones del AP.

Sin embargo, no veremos este paso en detalle, ya que es un procedimiento incómodo que genera GIC,s con un gran número de reglas, y no aporta nada necesario para razonamientos posteriores.

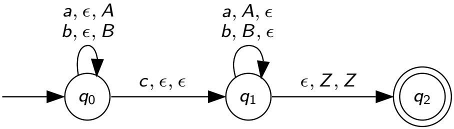
7 Autómatas de pila

Sabiendo que a partir de una GIC se puede obtener un AP equivalente, ahora la pregunta es:

- El lenguaje que acepta un AP, ¿es siempre un LIC?
- La respuesta es que sí, ya que dado un AP, se puede también obtener una GIC equivalente, cuyas reglas se calculan a partir de las transiciones del AP.

Sin embargo, no veremos este paso en detalle, ya que es un procedimiento incómodo que genera GIC,s con un gran número de reglas, y no aporta nada necesario para razonamientos posteriores.

Sí nos centraremos en el siguiente aspecto. Consideremos el lenguaje $\{wcw^I \mid w \in \{a, b\}^*\}$. Para este lenguaje, es posible construir un AP determinista:



Pero cualquier AP que acepte $\{ww^I \mid w \in \{a, b\}^*\}$ no puede ser determinista, ya que cada símbolo puede implicar: o bien su almacenamiento en la pila, o bien la comparación y el borrado.

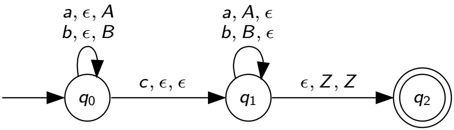
7 Autómatas de pila

Sabiendo que a partir de una GIC se puede obtener un AP equivalente, ahora la pregunta es:

- El lenguaje que acepta un AP, ¿es siempre un LIC?
- La respuesta es que sí, ya que dado un AP, se puede también obtener una GIC equivalente, cuyas reglas se calculan a partir de las transiciones del AP.

Sin embargo, no veremos este paso en detalle, ya que es un procedimiento incómodo que genera GIC,s con un gran número de reglas, y no aporta nada necesario para razonamientos posteriores.

Sí nos centraremos en el siguiente aspecto. Consideremos el lenguaje $\{wcw^I \mid w \in \{a, b\}^*\}$. Para este lenguaje, es posible construir un AP determinista:



Pero cualquier AP que acepte $\{ww^I \mid w \in \{a, b\}^*\}$ no puede ser determinista, ya que cada símbolo puede implicar: o bien su almacenamiento en la pila, o bien la comparación y el borrado.

Por tanto, al contrario de lo que ocurre en los lenguajes regulares, donde los AFN,s y los AFD,s son equivalentes, aquí no, y ocurre que los APN,s aceptan más lenguajes que los APD,s. Así pues, hemos alcanzado ya el siguiente grado de clasificación:

$$\textit{LR,s} \subset \textit{LIC,s deterministas} \subset \textit{LIC,s no deterministas}$$

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Demostración: A partir de un AP M_1 tal que $L(M_1) = L_1$ y de un AFD M_2 tal que $L(M_2) = L_2$, siempre se podrá construir un nuevo AP M que haga funcionar M_1 y M_2 en paralelo mediante operaciones de producto cartesiano, y tal que $L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2$. $\square$

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Demostración: A partir de un AP M_1 tal que $L(M_1) = L_1$ y de un AFD M_2 tal que $L(M_2) = L_2$, siempre se podrá construir un nuevo AP M que haga funcionar M_1 y M_2 en paralelo mediante operaciones de producto cartesiano, y tal que $L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2$. □

Algunos usos interesantes de este teorema son los siguientes.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{a^n b^n \mid n \geq 0 \text{ y } n \neq 100\}$?

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Demostración: A partir de un AP M_1 tal que $L(M_1) = L_1$ y de un AFD M_2 tal que $L(M_2) = L_2$, siempre se podrá construir un nuevo AP M que haga funcionar M_1 y M_2 en paralelo mediante operaciones de producto cartesiano, y tal que $L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2$. □

Algunos usos interesantes de este teorema son los siguientes.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{a^n b^n \mid n \geq 0 \text{ y } n \neq 100\}$?

Solución: Sí, ya que podemos escribir $L = L_1 \cap (L_3 - L_2)$, donde $L_1 = \{a^n b^n \mid n \geq 0\}$ es un LIC, $L_2 = \{a^{100} b^{100}\}$ es regular, $L_3 = \{a^* b^*\}$ es regular, y $L_3 - L_2$ es también regular.

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Demostración: A partir de un AP M_1 tal que $L(M_1) = L_1$ y de un AFD M_2 tal que $L(M_2) = L_2$, siempre se podrá construir un nuevo AP M que haga funcionar M_1 y M_2 en paralelo mediante operaciones de producto cartesiano, y tal que $L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2$. □

Algunos usos interesantes de este teorema son los siguientes.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{a^n b^n \mid n \geq 0 \text{ y } n \neq 100\}$?

Solución: Sí, ya que podemos escribir $L = L_1 \cap (L_3 - L_2)$, donde $L_1 = \{a^n b^n \mid n \geq 0\}$ es un LIC, $L_2 = \{a^{100} b^{100}\}$ es regular, $L_3 = \{a^* b^*\}$ es regular, y $L_3 - L_2$ es también regular.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{w \mid w \in \{a, b, c\}^*$ tal que el número de $a$es es igual al número de $b$es e igual al número de $c$es, sin importar el orden $\}$?

Teorema

Si L_1 es un LIC y L_2 es un lenguaje regular, entonces $L_1 \cap L_2$ es un LIC.

Demostración: A partir de un AP M_1 tal que $L(M_1) = L_1$ y de un AFD M_2 tal que $L(M_2) = L_2$, siempre se podrá construir un nuevo AP M que haga funcionar M_1 y M_2 en paralelo mediante operaciones de producto cartesiano, y tal que $L(M) = L(M_1) \cap L(M_2) = L_1 \cap L_2$. □

Algunos usos interesantes de este teorema son los siguientes.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{a^n b^n \mid n \geq 0 \text{ y } n \neq 100\}$?

Solución: Sí, ya que podemos escribir $L = L_1 \cap (L_3 - L_2)$, donde $L_1 = \{a^n b^n \mid n \geq 0\}$ es un LIC, $L_2 = \{a^{100} b^{100}\}$ es regular, $L_3 = \{a^* b^*\}$ es regular, y $L_3 - L_2$ es también regular.

Ejercicio

¿Es independiente del contexto el lenguaje $L = \{w \mid w \in \{a, b, c\}^*$ tal que el número de $a$es es igual al número de $b$es e igual al número de $c$es, sin importar el orden $\}$?

Solución: No, ya que podemos escribir $L_1 = L \cap L_2$, donde $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ no es un LIC, y $L_2 = a^* b^* c^*$ es regular. Si L fuera un LIC, L_1 debería serlo también y no lo es.

Contenidos

8 Forma normal de Greibach

La **forma normal de Greibach** o **FNG** restringe también las posiciones de los símbolos terminales y no terminales de las reglas de una GIC. Antes de introducirla, veremos dos resultados previos.

Teorema

Si $A \to \alpha B \gamma$ es una producción de una GIC y $B \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ son todas las posibles reescrituras del no terminal B, entonces la producción $A \to \alpha B \gamma$ se puede reemplazar por $A \to \alpha\beta_1\gamma \mid \alpha\beta_2\gamma \mid \ldots \mid \alpha\beta_m\gamma$ sin que varíe el lenguaje generado por la GIC. $\square$

8 Forma normal de Greibach

La **forma normal de Greibach** o **FNG** restringe también las posiciones de los símbolos terminales y no terminales de las reglas de una GIC. Antes de introducirla, veremos dos resultados previos.

Teorema

Si $A \to \alpha B \gamma$ es una producción de una GIC y $B \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ son todas las posibles reescrituras del no terminal B, entonces la producción $A \to \alpha B \gamma$ se puede reemplazar por $A \to \alpha\beta_1\gamma \mid \alpha\beta_2\gamma \mid \ldots \mid \alpha\beta_m\gamma$ sin que varíe el lenguaje generado por la GIC. □

Una regla de la forma $A \to \alpha A$ se denomina **recursiva por la derecha** y una regla de la forma $A \to A\alpha$ se denomina **recursiva por la izquierda**. Para muchas aplicaciones, es deseable que no exista recursividad por la izquierda. El siguiente resultado proporciona una forma de eliminarla.

Teorema

Dada una GIC donde $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n$ son todas las reglas de A recursivas por la izquierda, y $A \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ las restantes reescrituras de A, se puede construir una GIC equivalente introduciendo un no terminal Z y reemplazando todas las reglas precedentes por:

$$
\begin{aligned}
A &\to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \mid \beta_1 Z \mid \beta_2 Z \mid \ldots \mid \beta_m Z \\
Z &\to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \mid \alpha_1 Z \mid \alpha_2 Z \mid \ldots \mid \alpha_n Z
\end{aligned}
$$

8 Forma normal de Greibach

La **forma normal de Greibach** o **FNG** restringe también las posiciones de los símbolos terminales y no terminales de las reglas de una GIC. Antes de introducirla, veremos dos resultados previos.

Teorema

Si $A \rightarrow \alpha B \gamma$ es una producción de una GIC y $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ son todas las posibles reescrituras del no terminal B, entonces la producción $A \rightarrow \alpha B \gamma$ se puede reemplazar por $A \rightarrow \alpha \beta_1 \gamma \mid \alpha \beta_2 \gamma \mid \ldots \mid \alpha \beta_m \gamma$ sin que varíe el lenguaje generado por la GIC. □

Una regla de la forma $A \rightarrow \alpha A$ se denomina **recursiva por la derecha** y una regla de la forma $A \rightarrow A\alpha$ se denomina **recursiva por la izquierda**. Para muchas aplicaciones, es deseable que no exista recursividad por la izquierda. El siguiente resultado proporciona una forma de eliminarla.

Teorema

Dada una GIC donde $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n$ son todas las reglas de A recursivas por la izquierda, y $A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ las restantes reescrituras de A, se puede construir una GIC equivalente introduciendo un no terminal Z y reemplazando todas las reglas precedentes por:

$$\begin{array}{l} A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \mid \beta_1 Z \mid \beta_2 Z \mid \ldots \mid \beta_m Z \\ Z \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n \mid \alpha_1 Z \mid \alpha_2 Z \mid \ldots \mid \alpha_n Z \end{array}$$

Demostración: Obsérvese que, en ambos casos, las cadenas derivadas de A constituyen el lenguaje regular $\{\beta_1 \mid \beta_2 \mid \ldots \mid \beta_m\} \{\alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n\}^*$. □

Ejercicio

Dada la siguiente GIC, construya otra GIC equivalente sin recursividades por la izquierda:

$$\begin{array}{l} S \rightarrow Sa \mid Sb \mid cA \\ A \rightarrow Aa \mid a \mid \epsilon \end{array}$$

Ejercicio

Dada la siguiente GIC, construya otra GIC equivalente sin recursividades por la izquierda:

$$\begin{array}{l} S \rightarrow Sa \mid Sb \mid cA \\ A \rightarrow Aa \mid a \mid \epsilon \end{array}$$

Solución:

$$\begin{array}{l} S \rightarrow cA \mid cAZ_1 \\ Z_1 \rightarrow a \mid b \mid aZ_1 \mid bZ_1 \\ A \rightarrow a \mid \epsilon \mid aZ_2 \mid Z_2 \\ Z_2 \rightarrow a \mid aZ_2 \end{array}$$

Obsérvese que al eliminar las reglas recursivas por la izquierda aparecen nuevos símbolos no terminales y reglas recursivas por la derecha.

8 Forma normal de Greibach

Ejercicio

Dada la siguiente GIC, construya otra GIC equivalente sin recursividades por la izquierda:

$$\begin{aligned} S &\rightarrow Sa \mid Sb \mid cA \\ A &\rightarrow Aa \mid a \mid \epsilon \end{aligned}$$

Solución:

$$\begin{aligned} S &\rightarrow cA \mid cAZ_1 \\ Z_1 &\rightarrow a \mid b \mid aZ_1 \mid bZ_1 \\ A &\rightarrow a \mid \epsilon \mid aZ_2 \mid Z_2 \\ Z_2 &\rightarrow a \mid aZ_2 \end{aligned}$$

Obsérvese que al eliminar las reglas recursivas por la izquierda aparecen nuevos símbolos no terminales y reglas recursivas por la derecha.

Definición

Una GIC está en **forma normal de Greibach** o **FNG** si todas las reglas son de la forma $A \rightarrow a\alpha$, donde a es un terminal y $\alpha \in N^*$.

Obsérvese que la presencia de un símbolo terminal al principio de la parte derecha de cada regla evita la existencia de recursividades por la izquierda y de reglas ϵ (aunque siempre podremos incluir $S \rightarrow \epsilon$ cuando ϵ pertenezca al lenguaje de la gramática).

8 Forma normal de Greibach

Para cualquier GIC siempre es posible construir otra equivalente en FNG.

El algoritmo consta de las siguientes etapas:

1. Sea $G = (N, \Sigma, P, S)$ en FNC, y supongamos también que $N = \{A_1, A_2, \ldots, A_n\}$, donde $A_1 = S$. Entonces modificamos las reglas de P de forma que si $A_r \to A_s\alpha \in P$, $r < s$.

   Supongamos que hemos modificado ya algunas reglas, de forma que para $1 \leq i \leq k$, si $A_i \to A_j\alpha$, entonces $i < j$. Mostraremos ahora cómo modificar las reglas para A_{k+1}. Si tenemos que $A_{k+1} \to A_j\alpha$ es una regla con $k + 1 > j$, generamos un nuevo conjunto de reglas para reemplazar A_j por el lado derecho de todas las reglas de la forma $A_j \to \beta$, tal y como indica el primero de los teoremas visto en esta sección.

   Al realizar estas substituciones, obtendremos reglas de la forma $A_{k+1} \to A_r\gamma$, donde:

   - Si $k + 1 < r$, ya tenemos una regla de la forma deseada.
   - Si $k + 1 > r$, debemos repetir el proceso. Dado que sólo hay k índices menores que $k + 1$, después de repetir el proceso $k - 1$ veces, obtendremos reglas donde $k + 1 \leq r$.
   - Si $k + 1 = r$, tendremos una producción recursiva por la izquierda que puede ser eliminada mediante el método que se propone en el segundo teorema de esta sección, introduciendo un nuevo no terminal Z_{k+1}.

2. Después de repetir este proceso para todos los no terminales originales $A_1, A_2, \ldots, A_n$, sólo tendremos reglas de alguna de estas tres formas:
   - $A_k \to A_j\alpha$, con $k < j$.
   - $A_k \to a\alpha$, con $a \in \Sigma$.
   - $Z_k \to \alpha$, con $\alpha \in (N \cup \{Z_1, Z_2, \ldots, Z_n\})^*$.

   Puesto que A_n es el no terminal de mayor índice, todas sus reescrituras han de ser del segundo tipo, es decir, empiezan por un terminal.

   Todas las reescrituras de A_{n-1} comenzarán con un terminal, o con A_n, el cual puede ser reemplazado según el primer teorema, pasando todas las reglas a comenzar también con un terminal.

   Este proceso se repite para $A_{n-2}, A_{n-3}, \ldots, A_1$, hasta que todas las reescrituras de los no terminales originales comiencen con un terminal.

8 Forma normal de Greibach

2. Después de repetir este proceso para todos los no terminales originales $A_1, A_2, \ldots, A_n$, sólo tendremos reglas de alguna de estas tres formas:
   - $A_k \to A_j\alpha$, con $k < j$.
   - $A_k \to a\alpha$, con $a \in \Sigma$.
   - $Z_k \to \alpha$, con $\alpha \in (N \cup \{Z_1, Z_2, \ldots, Z_n\})^*$.

   Puesto que A_n es el no terminal de mayor índice, todas sus reescrituras han de ser del segundo tipo, es decir, empiezan por un terminal.

   Todas las reescrituras de A_{n-1} comenzarán con un terminal, o con A_n, el cual puede ser reemplazado según el primer teorema, pasando todas las reglas a comenzar también con un terminal.

   Este proceso se repite para $A_{n-2}, A_{n-3}, \ldots, A_1$, hasta que todas las reescrituras de los no terminales originales comiencen con un terminal.

3. Consideramos ahora las reglas de $Z_1, Z_2, \ldots, Z_n$. Dado que la gramática original estaba en forma normal de Chomsky, y sólo hemos usado los teoremas vistos en esta sección, ninguna regla $Z_i \to \alpha$ tendrá otro Z_j como primer símbolo de α.

   Por tanto, α comenzará ya con un terminal, o con un no terminal A_k, el cual puede ser substituido según el primer teorema, pasando todas las reglas a estar en la forma deseada.

Ejercicio

Dada la siguiente GIC, construya otra equivalente en forma normal de Greibach:

$$A_1 \to A_2A_2 \mid a \qquad\qquad A_2 \to A_1A_2 \mid b$$

Ejercicio

Dada la siguiente GIC, construya otra equivalente en forma normal de Greibach:

$$A_1 \to A_2A_2 \mid a \qquad A_2 \to A_1A_2 \mid b$$

Solución:

La regla $A_1 \to A_2A_2$ ya está bien para la primera etapa. Las reglas $A_1 \to a$ y $A_2 \to b$ ya están en la forma deseada. La regla $A_2 \to A_1A_2$ no está bien. Aplicamos sobre ella el primer teorema y obtenemos:

$$A_1 \to A_2A_2 \mid a \qquad A_2 \to A_2A_2A_2 \mid aA_2 \mid b$$

Mediante el segundo teorema, eliminamos ahora la recursividad izquierda de A_2 y obtenemos:

$$A_1 \to A_2A_2 \mid a \qquad A_2 \to aA_2 \mid aA_2Z \mid b \mid bZ \qquad Z \to A_2A_2 \mid A_2A_2Z$$

Y finalmente, substituimos A_2 de forma apropiada para que todas las reescrituras comiencen con un terminal:

$$A_1 \to aA_2A_2 \mid aA_2ZA_2 \mid bA_2 \mid bZA_2 \mid a$$
$$A_2 \to aA_2 \mid aA_2Z \mid b \mid bZ$$
$$Z \to aA_2A_2 \mid aA_2ZA_2 \mid bA_2 \mid bZA_2 \mid aA_2A_2Z \mid aA_2ZA_2Z \mid bA_2Z \mid bZA_2Z$$

8 Forma normal de Greibach

La forma normal de Greibach es interesante por varias razones:

- En primer lugar, dado que el uso de una producción introduce exactamente un solo símbolo terminal, una cadena de longitud n tiene una derivación de exactamente n pasos.
- Además, al analizar sintácticamente una cadena, si la gramática correspondiente está en FNG, el *parser* puede guiarse comparando los símbolos de la cadena con los terminales que encabezan las reescrituras, eliminando así parcial o totalmente el no determinismo inherente a la tarea de análisis sintáctico.
- De igual forma, si se construye un AP a partir de una gramática en FNG, este AP también podrá realizar procesos de reconocimiento guiados, comparando los símbolos de la cadena de entrada con los terminales que encabezan las etiquetas de las transiciones (es lo que se conoce como técnica de *look-ahead*).
- Por último, queda garantizado también que dicho AP no tendrá ϵ-transiciones, lo que demuestra que siempre es posible eliminar este tipo de transiciones de cualquier autómata de pila.

Fin del capítulo

"Lenguajes Independientes del Contexto y Autómatas de Pila"