Teoría de Lenguajes
Segunda parte
Teórica 8: Reescritura de gramáticas

Verónica Becher

Segundo cuatrimestre 2020

Bibliografía para esta clase:
[1] A. V. Aho, J. D. Ullman, The Theory of Parsing, Translation, and Compiling, Vol. 1 , Parsing. Prentice Hall, 1972. `https://www-2.dc.uba.ar/staff/becher/Aho-Ullman-Parsing-V1.pdf` Capítulo 2.4.

Nuestro problema
Dada una expresión en un lenguaje formal, tales como,

```
void main(int n) { char c; while () .... return c; }
```

$x^2 + y^2 = z^2$

$p \vee q \wedge (\neg r)$

¿Qué debe hacer una computadora para "entenderla"?

Nuestra materia da respuestas a esta pregunta.

Respuesta (anticipo de lo que veremos)
Dada una *expresión* en un *lenguaje formal*,

¿Qué debe hacer una *computadora* para "entenderla"?

1. Revisar que la expresión tenga símbolos del *alfabeto* solamente.
2. Revisar que la expresión tenga una forma que respete la *gramática*.
3. Mapear la expresión, usando la gramática, a una estructura de datos (*árbol de derivación con atributos*).

Lo escrito en itálica requiere definición.

Antes de avanzar ...
¿Todos los lenguajes formales admiten este análisis?

¿Cuál es la dificultad computacional de hacer este análisis?

Lenguajes y complejidad computacional

Hay lenguajes que se analizan en tiempo lineal en el tamaño de la entrada (esta es la mínima complejidad posible cuando hay que leer la entrada). Estos son los lenguajes regulares y algunos lenguajes libres de contexto.

El análisis de los demás lenguajes libres de contexto usa tiempo polinomial en el tamaño de la entrada. Y para lenguajes arbitrarios no hay cota en la complejidad computacional de analizarlos.

Recordemos las inclusiones conocidas de teoría de complejidad

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$$

L:	espacio logaritimico en máquina Turing deterministica
NL:	espacio logaritmico en máquina Turing no deterministica
P:	tiempo polinomial en máquina Turing deterministica
NP:	tiempo polinomial en una máquina Turing no deterministica
PSPACE:	espacio polinomial en máquina Turing deterministica

Teorema de Savitch's muestra que PSPACE = NPSPACE

Lenguajes y complejidad computacional

Hay lenguajes matemáticamente definidos que *no* pueden ser analizados por ninguna computadora (máquina de Turing):

por ejemplo, los programas en C que se cuelgan

(tema visto en la materia Lógica y Computabilidad).

Los lenguajes que no son recursivamente enumerables no pueden ser reconocidos por ningna máquina de Turing.

Alfabeto, Cadena, Lenguaje

Recordemos que un *alfabeto* es un conjunto finito de símbolos (elemento). Por ejemplo, $\{0, 1\}$, $\{a, b, c, \ldots, x, y, z\}$, $\{(,), *, -, +, id\}$.

Una *cadena* es una secuencia finita de símbolos. Por ejemplo, 01010.

Un *lenguaje* es un conjunto de cadenas. Por ejemplo, el conjunto vacío, el conjunto de las cadenas de dos símbolos, el conjunto de palíndromos.

Hay lenguajes naturales (quedan todos fuera del alcance de esta materia) y lenguajes formales. Los lenguajes formales se definen matemáticamente, (mediante gramáticas y mediante autómatas, ver la jerarquía de Chomsky en la diapositivas siguientes).

En esta materia nos concentramos en los lenguajes regulares y en los lenguajes libres de contexto.

Gramática

Definición 1. *Una gramática formal es una cuádrupla $G = (N, T, S, P)$ donde:*

N es un conjunto finito de símbolos no terminales (variables).

T es un conjunto finito de símbolos terminales (constantes).

S es un símbolo distinguido de N, el símbolo inicial.

P es un conjunto finito de reglas de producción, cada una de la forma $(N \cup T)^* \to (N \cup T)^*$ donde $*$ es la clausura de Kleene.

Ejemplo 2. *$G = (N, T, S, P)$ con $N = \{E\}$, $T = \{(,), id, +, -, *\}$, $S = E$ y P es*

$$\begin{aligned} E &\to E + E \\ E &\to -E \\ E &\to E * E \\ E &\to (E) \\ E &\to id \end{aligned}$$

Si hay muchas producciones con el mismo lado izquierdo, podemos escribir $E \to E + E \mid E * E \mid (E) \mid -E \mid id$

Lenguaje generado por una gramática

Definición 3 (Derivación $\Rightarrow$)**.** *Sea $G = (N, T, P, S)$ una gramática.*
Si $A \in N$, $A \to \gamma \in P$ y $\alpha A \beta \in (N \cup T)^*$ entonces

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

La relación $\Rightarrow$ es un subconjunto $(N \cup T)^* \times (N \cup T)^*$ y significa derivar en un solo paso.

Las relaciones $\overset{+}{\Rightarrow}$ y $\overset{*}{\Rightarrow}$ son la clausura transitiva y reflexo-transitiva respectivamente, (más de un paso de derivación, y cero o más pasos).

Ejemplo de derivación

En cada paso de la de la derivación debemos elegir qué símbolo no-terminal reescribiremos y luego debemos elegir una producción que tenga ese símbolo del lado izquierdo.

Usando la gramática $E \to E + E \mid E * E \mid (E) \mid -E \mid id$.

Si elegimos el no-terminal más a la izquierda, escribimos $\underset{L}{\Rightarrow}$ ‘

$E \overset{*}{\underset{L}{\Rightarrow}} -(id)$, porque $E \underset{L}{\Rightarrow} -E \underset{L}{\Rightarrow} -(E) \underset{L}{\Rightarrow} -(id)$

$E \overset{*}{\underset{L}{\Rightarrow}} (id + id)$ porque $E \underset{L}{\Rightarrow} (E) \underset{L}{\Rightarrow} (E + E) \underset{L}{\Rightarrow} (id + E) \underset{L}{\Rightarrow} (id + id)$.

Si elegimos el no termnal más a la derecha, escribimos $\underset{R}{\Rightarrow}$.

$E \underset{R}{\Rightarrow} (E) \underset{R}{\Rightarrow} (E + E) \underset{R}{\Rightarrow} (E + id) \underset{R}{\Rightarrow} (id + id)$.

Lenguaje generado de una gramática G

Si $\alpha \in (N \cup T)^*$ y $S \overset{*}{\Rightarrow} \alpha$ decimos que α es una *forma sentencial* de G.

Definición 4 (Lenguaje generado de una gramática G)**.** *Dada una gramática $G = (N, T, P, S)$,*

$$L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$$

Árbol de derivación

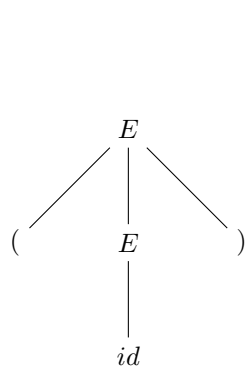
Asumamos una gramática $G = (N, T, P, S)$.

Definición 5 (Árbol de derivación)**.** *Un árbol de derivación es una representación gráfica de una derivación que* no *muestra el orden en que sea aplicaron las producciones. Las etiquetas de los nodos están en $N \cup T \cup \{\lambda\}$. Las de los nodos internos son no terminales y las de sus hijos son los símbolos del cuerpo de una producción (lado derecho).*

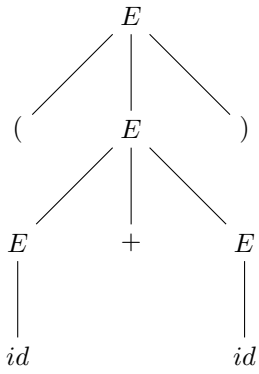
Si un nodo tiene etiqueta A entonces hay una producción $A \to X_1 \dots X_n$ y el nodo tiene n descendientes, etiquetados $X_1, X_2, \dots, X_n$.

Cada árbol de derivación tiene asociada una única derivación más a la izquierda, y una única derivación más a la derecha.

Ejemplo árbol de derivación y su derivación más a la iquierda



$E \underset{L}{\Rightarrow} (E) \underset{L}{\Rightarrow} (id)$

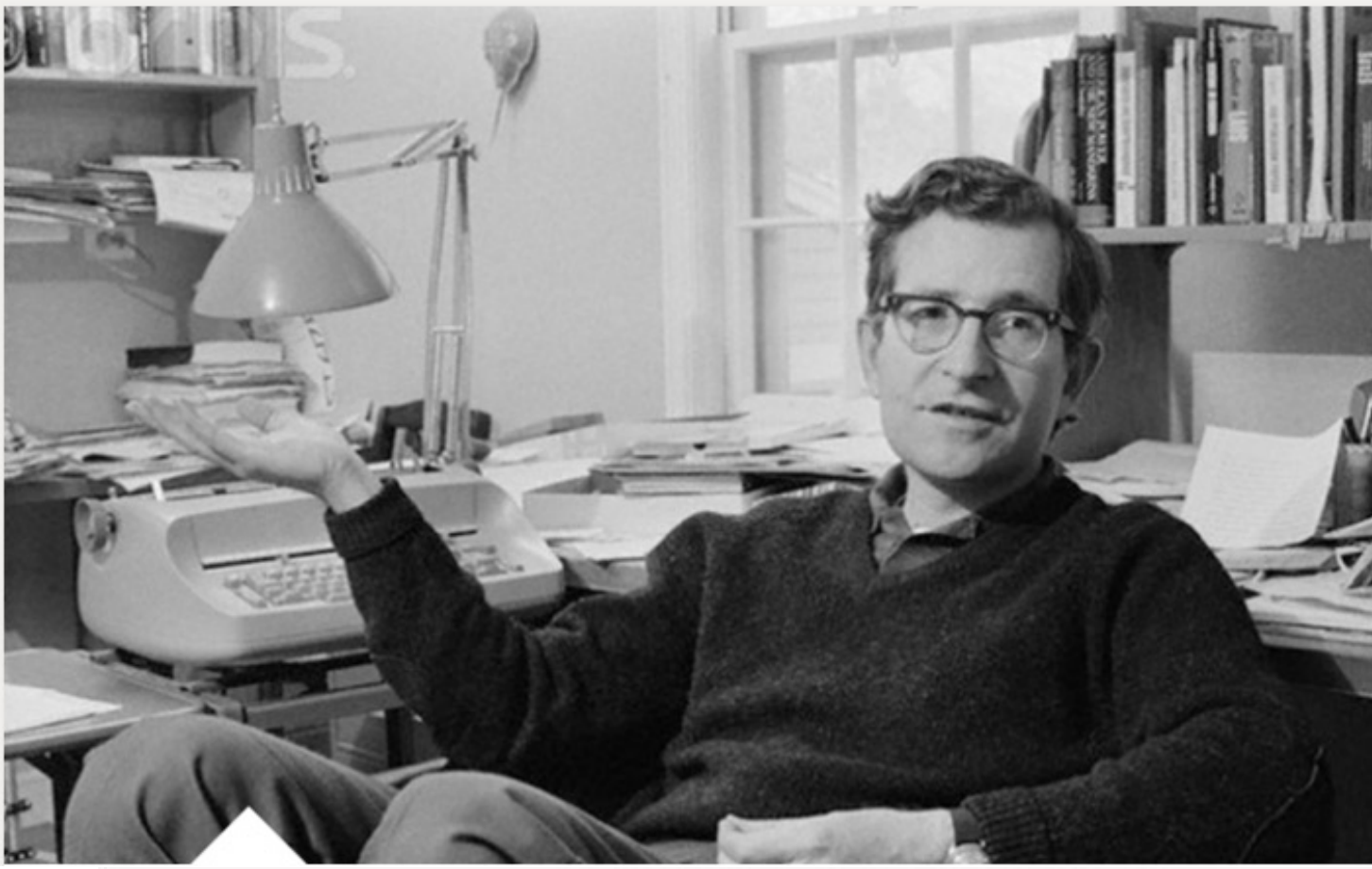


$E \underset{L}{\Rightarrow} (E) \underset{L}{\Rightarrow} (E+E) \underset{L}{\Rightarrow} (id+E) \underset{L}{\Rightarrow} (id+id)$

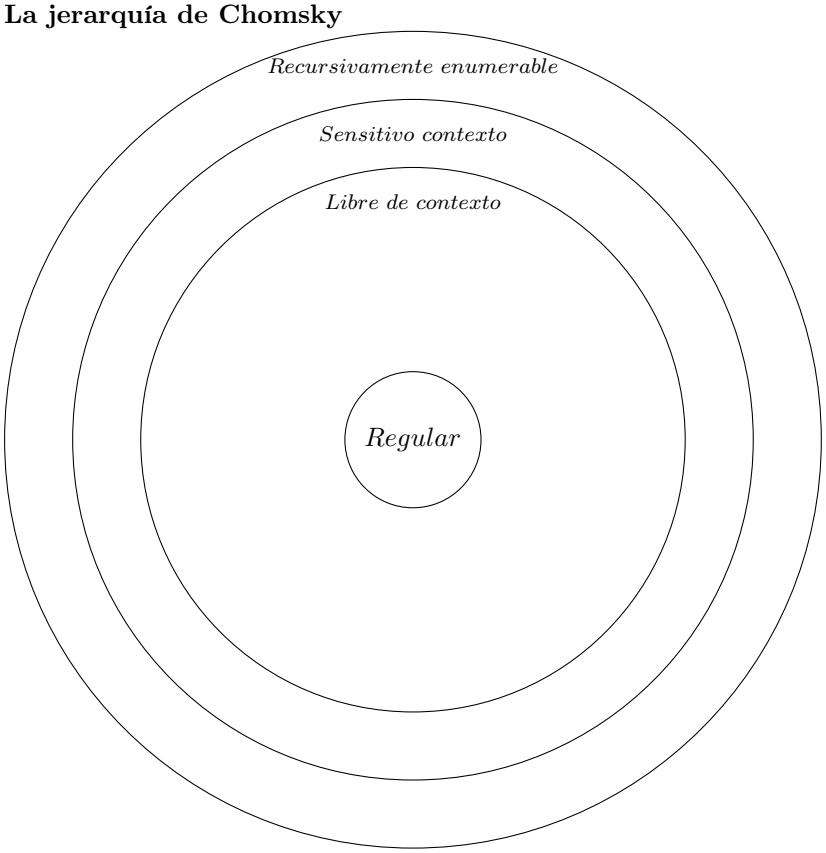
La jerarquía de Chomsky (Noam Chomsky en 1956).

Es una clasificación jerárquica de tipos de gramáticas formales que generan lenguajes formales.

Noam Chomsky



La jerarquía de Chomsky



La jerarquía de Chomsky (Noam Chomsky en 1956).

Gramáticas de tipo 0 (gramáticas sin restricciones).

Incluyen a todas las gramáticas formales. Estas gramáticas generan todos los lenguajes aceptados por una máquina de Turing. Se los llama *lenguajes recursivamente enumerables* (r.e).

Ejemplo: La gramática $X \to Y$, con X cadena de terminales y no terminales e Y cadena de terminales y no terminales o vacía.

Un lenguaje r.e. es el conjunto de programas en C que terminan.

La jerarquía de Chomsky (Noam Chomsky en 1956).

Gramáticas de tipo 1 (gramáticas sensibles al contexto).

Las reglas son de la forma $\alpha \to \beta$, con $|\alpha| \leq |\beta|$ α, β cadenas de terminales y no terminales. La regla $S \to \lambda$ está permitida si S no aparece en la parte derecha de ninguna regla.

Estas gramáticas generan todos los lenguajes aceptados por los autómatas linealmente acotados, que son máquinas de Turing deterministicas cuya cinta de memoria está acotada por un múltiplo de la longitud de entrada. Se los llama *lenguajes sensibles al contexto*.

Ejemplo

$$\begin{aligned} S &\to abc|aSBc \\ cB &\to Bc \\ bB &\to bb \end{aligned}$$

genera el lenguaje $\{a^n b^n c^n : n \geq 1\}$, que no es libre de contexto.

Reconocer un lenguaje generado por una gramática sensitiva al contexto es PSPACE-completo.

La jerarquía de Chomsky (Noam Chomsky en 1956).

Gramáticas de tipo 2 (gramáticas libres de contexto).

Las reglas son de la forma $A \to \gamma$ con A un no terminal y γ una cadena de terminales y no terminales. Estas gramáticas generan todos los lenguajes aceptados por un autómata de pila. Se los llama *lenguajes libres de contexto*.

Ejemplo. La gramática $S \to aSb|\lambda$ genera el lenguaje $\{a^n b^n : n \geq 0\}$, que no es regular.

Gramática libre de contexto

Como ya sabemos, no es cierto que para cada autómata de pila no determinístico exista uno determinístico que reconozca el mismo lenguaje.

Los lenguajes libres de contexto aceptados por los autómatas de pila determinísticos se llaman lenguajes LR, y su máyor interés es que se los reconce en tiempo lineal en el tamaño de la entrada. Los estudiaremos.

Un subconjunto propio de los lenguajes LR es el de los lenguajes LL.

Son los generados por gramáticas LL y el algoritmo para reconocerlos es muy sencillo. Lo estudiaremos.

Para reconocer un lenguaje libre de contexto no determinístico hay un algoritmo de complejidad cúbica en el tamaño de la entrada. Estudiaremos el algoritmo CYK.

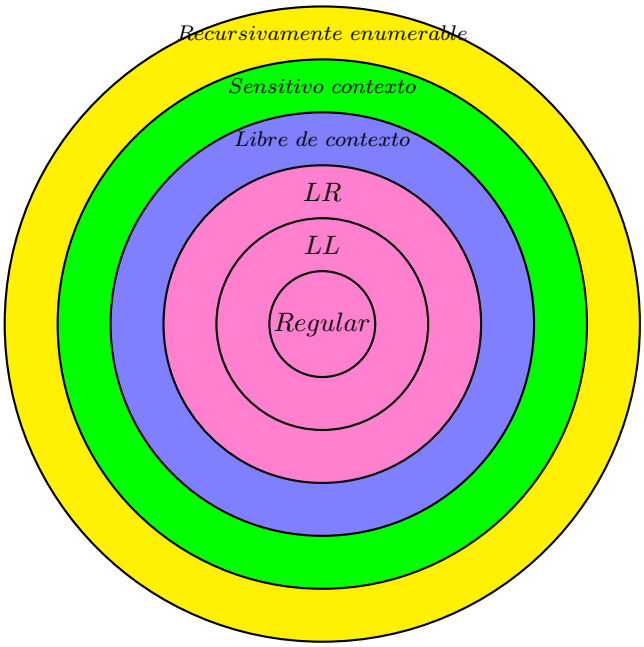
La jerarquía de Chomsky (Noam Chomsky en 1956).

Gramáticas de tipo 3 (gramáticas regulares). Las reglas son de la forma $A \to a$, $A \to aB$ con A, B símbolos no terminales y a un terminal. Alternativamente las reglas son de la forma $A \to a$ y $A \to Ba$ con A, B símbolos no terminales y a un terminal. La regla $S \to \lambda$ también está permitida si S no aparece en la parte derecha de ninguna regla. Estas gramáticas generan todos los lenguajes aceptados por autómatas finitos, deterministicos o no deterministicos. Se los llama *lenguajes regulares*. También los lenguajes regulares se pueden definir mediante *expresiones regulares*.

Ejemplo. $A \to aA|a$ genera el lenguaje regular a^+.

Para determinar si una cadena pertenece o no o aun lenguaje regular, siempre hay un algoritmo lineal en el tamaño de la cadena. Resulta de implementar la función de transición del autómata finito y simular las transisiones hasta determinar si se llega o no a un estado final.

La jearquía de Chomsky ampliada



Análisis Sintáctico o "parsing"

Definición 6 (Análisis Sintáctico o "parsing")**.** *Sea $G = (N, T, P, S)$ una gramática libre de contexto. Sea $\alpha \in (T \cup N)^*$. Un parsing más a la izquierda de α es la secuencia de producciones usadas en la derivación más a la izquierda de α desde S . Un parsing más a la derecha de α es la secuencia de producciones usadas en la derivación más a la derecha de α desde S.*

Definición 7. *Una gramática G es* abigua *si hay una expresión de $L(G)$ que tiene más de una derivación más a la izquierda, o más de una derivación más a la derecha.*

Ejemplo: Sea G la gramática con producciones

$$E \to E + E \mid E * E \mid (E) \mid -E \mid id$$

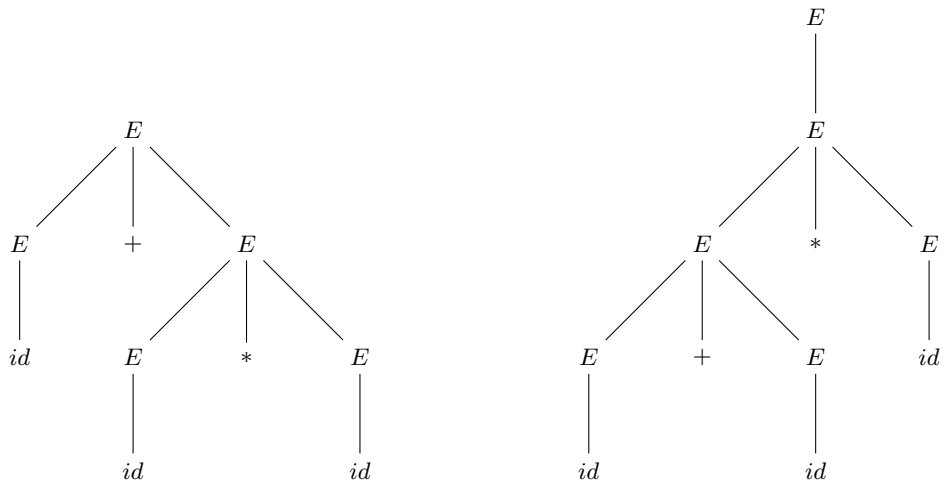
Para $id + id * id$ de $L(G)$ hay dos derivaciones más a la izquierda

$$E \underset{L}{\Rightarrow} E + E \underset{L}{\Rightarrow} id + E \underset{L}{\Rightarrow} id + E * E \underset{L}{\Rightarrow} id + id * E \underset{L}{\Rightarrow} id + id * id$$

y

$$E \underset{L}{\Rightarrow} E * E \underset{L}{\Rightarrow} E + E * E \underset{L}{\Rightarrow} id + E * E \underset{L}{\Rightarrow} id + id * E \underset{L}{\Rightarrow} id + id * id$$

La expresión $id + id * id$ tiene dos árboles de derivación en esta gramática.



Gramática ambigua y lenguajes ambigos

Definición 8 (Gramática no ambigua)**.** *Una gramática libre de contexto es* no ambigua *cada cadena del lenguaje tiene una única derivación a la izquierda.*

Un subconjunto propio de las gramáticas libres de contexto no-ambiguas son las libres del contexto determinísticas.

Recordemos (de Lógica y Computabilidad): un problema que requiere respuesta por sí o por no, *no es decidible* cuando no hay ningún algoritmo capaz de responder todas las (infinitas) instancias del problema.

El problema de decidir si una gramática libre de contexto es ambigua no es decible.

Gramática ambigua y lenguajes ambigos

Cualquier lenguaje no vacío admite una gramática ambigua al tomar una gramática no ambigua e introducir una regla duplicada (el único lenguaje sin gramáticas ambiguas es el lenguaje vacío).

Hay lenguajes que solamente gramáticas ambiguas. Se los llama *inherentemente ambiguos*. Su existencia fue demostrada por Rohit Parikh en 1961. Un ejemplo de un lenguaje inherentemente ambiguo es

$$\{a^n b^m c^m d^n | n, m > 0\} \cup \{a^n b^n c^m d^m | n, m > 0\}$$

Este conjunto es libre del contexto, dado que la unión de dos lenguajes libres del contexto es siempre libre del contexto.

Gramáticas ambiguas y lenguajes ambiguos

Si nos enfrentamos con una gramática ambigua, podemos hacer estos intentos:

1. Cambiar la gramática
2. Descartar árboles de derivación, dando reglas de precedencia

Gramáticas recursivas a izquierda

Definición 9 (Gramáticas recursivas a izquierda)**.** *Una gramática $G = (N, T, P, S)$ libre de contexto es* recursiva a izquierda *si tiene un no termnal A tal que para alguna expresión $\alpha \in (N \cup T)^*$*

$$A \stackrel{+}{\Rightarrow} A\alpha$$

Se llama recursión inmediata cuando hay una producción $A \to A\alpha$.
Ejemplo. Consideremos la gramática que tiene estas producciones

$$E \to E + E \mid E * E \mid (E) \mid id$$

tenemos este ejemplo de recursión inmediata, $E \Rightarrow E + E$.

Otro ejemplo. Consideremos esta otra gramática con estas producciones

$$A \to BC \mid a$$
$$B \to CA \mid Ab$$

Un ejemplo de recursión no inmediata es $A \Rightarrow BC \Rightarrow AbC$.

Gramáticas recursivas a izquierda

Teorema 10 (Teorema 2.18 [1])**.** *Todo lenguaje libre de contexto tiene una gramática que no es recursiva a izquierda.*

Para demostrar este teorema daremos un algoritmo que transforma una gramática libre de contexto G dada, que es recursiva a izquierda, en otra gramática libre de contexto G' que no es recursiva a izquierda y $L(G) = L(G')$.

Algoritmo de eliminación de la recursion inmediata

Lemma 2.15 [1] (página 154)

Sea $G = (N, T, P, S)$ libre de contexto, donde P es el conjunto de producciones

$$A \to A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$$

donde ninguno de los β empieza con A.

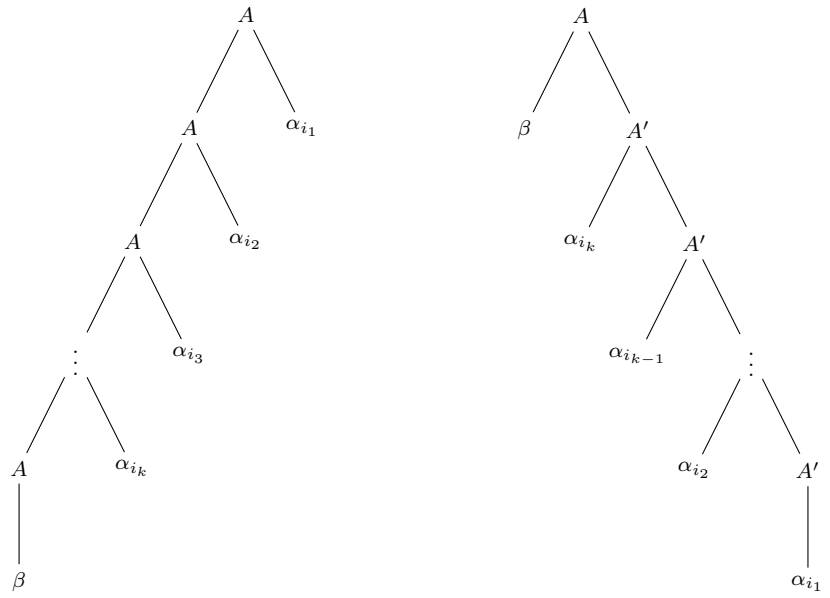
Sea $G' = (N \cup \{A'\}, T, P', S)$ con A' un nuevo símbolo no terminal y P idéntico a P pero reemplazando las producciones de lado izquierdo A por

$$A \to \beta_1 \mid \ldots \mid \beta_n \mid \beta_1 A' \mid \ldots \mid \beta_n A'$$
$$A' \to \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 A' \mid \ldots \mid \alpha_m A'$$

Notar P' tiene el doble de producciones que P.

Algoritmo de eliminación de la recursion inmediata

Arboles de derivación en G y en G' de $A \stackrel{*}{\Rightarrow} \beta\alpha_{i_k}\alpha_{i_{k-1}}\ldots\alpha_{i_2}\alpha_{i_1}$



Complejidad de la eliminación de la recursión inmediata

Notar P' tiene el doble de producciones que P.

El algoritmo transforma G en G', duplicando las producciones.

Tiene complejidad lineal en $|P|$.

Ejemplo de eliminación de recursión inmediata

Sea G la gramatica con producciones

$$\begin{aligned} E &\to E + T|T \\ T &\to T * F|F \\ F &\to (E)|a \end{aligned}$$

El algoritmo obtiene G' con E' y t' son simbolos no terminales nuevos y las producciones

$$\begin{aligned} E &\to T|TE' \\ E' &\to +T|TE' \end{aligned}$$

$$\begin{aligned} T &\rightarrow F|FT' \\ T' &\rightarrow *F|*FT' \\ F &\rightarrow (E)|a \end{aligned}$$

Algoritmo de eliminación de la recursion

Definición 11 (Gramática sin ciclos)**.** *Una gramática no tiene ciclos si para todo símbolo no termina A no hay derivaciones $A \stackrel{*}{\Rightarrow} A$.*

En las gramáticas libre de contexto los ciclos se originan en la existencia de producciones $A \rightarrow B$ con B un único no-terminal y en las producciones $A \rightarrow \lambda$.

Es posible transformar la gramática en otra sin ciclos eliminando las producciones $A \rightarrow \lambda$ (solamente podemos dejar $S \rightarrow \lambda$) y eliminando las producciones con un solo no-terminal en el cuerpo.

Ver Algoritmos 2.10 y 2.11 [Aho Ullman vol. 1].

Algoritmo de eliminación de la recursion

Algoritmo 2.13 [Aho Ullman vol 1] (página 155)

Input: $G = (N, T, P, S)$ libre de contexto, sin producciones λ, sin símbolos inútiles, y sin ciclos.
Output: G' libre de contexto, sin recursión a izquierda, $L(G) = L(G')$.

Sea $n = |N|$. Numerar los símbolos no terminales $A_1, A_2, \ldots, A_n$ Modificar G hasta obtener G' donde para cada producción $A_i \rightarrow \alpha$,

α empieza con un terminal o con un no terminal A_k con $k > i$:

`para` $i = 1$ `hasta` n `para` $j = 1$ `hasta` $i - 1$ `si (`$A_i \rightarrow A_j\gamma$ `y` $A_j \rightarrow \delta_1|\delta_2|\ldots|\delta_m$`) entonces` `Reemplazar` $A_i \rightarrow A_j\gamma$ `por` $A_i \rightarrow \delta_1\gamma|\delta_2\gamma|\ldots|\delta_m\gamma$. `Eliminar la recursion inmediata de` A_i

Notar que en el peor caso G' tiene esta cantidad de producciones

$$c_1 \ + \ c_1 \times c_2 \ + \ c_1 \times c_2 \times c_3 \ + \ \ldots + c_1 \times c_2 \times \ldots \times c_n$$

donde c_i es la cantidad de producciones de P con cabeza A_i.

En el peor caso cada $c_i = |P|/n$, entonces $|G'|$ tiene $O((|P|/n)^n)$ producciones (creció exponencialmente en la cantidad de símbolos no terminales de G).

Algoritmo de eliminación de la recursion

Ejemplo. Sea $G = (N, T, P, S)$

$$\begin{aligned} A &\rightarrow BC|a \\ B &\rightarrow CA|Ab \\ C &\rightarrow a \end{aligned}$$

Corremos el agoritmo $i = 1$ Sin cambios. $i = 2$ $B \rightarrow Ab$ es reemplazado por $B \rightarrow BCb|ab$.

La eliminacion de recursion inmediata de $B \rightarrow BCb|ab|CA$ deja

$$\begin{aligned} B &\rightarrow ab|CA|abB'|CAB' \\ B' &\rightarrow Cb|CbB' \end{aligned}$$

$i = 3$ sin cambios. Queda

$$\begin{aligned} A &\rightarrow BC|a \\ B &\rightarrow ab|CA|abB'|CAB' \\ B' &\rightarrow Cb|CbB' \\ C &\rightarrow a \end{aligned}$$

Transformaciones de gramáticas

Sección 2.4.2. [Aho Ullman vol 1] (página 143)

- Remover símbolos y producciones inútiles
- Eliminar producciones con λ excepto $S \to \lambda$
- Eliminar producciones cuyo cuerpo es un único no terminal
- Eliminar ciclos
- Eliminar recursióna izquierda
- Llevar la gramática a forma normal de Chomsky donde cada producción es de la forma $A \to BC$, y $A \to a$ con A, B, C no terminales y a terminal. Y en caso de que $\lambda \in L(G)$, $S \to \lambda$ y λ no aparece en otro lado.
- Llevar la gramática a forma normal de Greibach, donde todas las producciones son de la forma $A \to a\alpha$, y no hay producciones λ.

— ¿Cuándo habremos de vernos, con el trueno, otra vez, con el rayo o la lluvia, reunidas las tres? — Cuando el caos acabe, al fin de la batalla, bien se pierda o se gane. — Antes que el sol se ponga. ¿Y el lugar? — En el páramo.