

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

APUNTE IIC2223

Teoría de Autómatas y Lenguajes Formales

Autor
Cristóbal ROJAS

En base a apuntes de
Prof. Cristian RIVEROS

2 de diciembre de 2022



Índice

1. Lenguajes regulares	3
1.1. Palabras y autómatas	3
1.1.1. Palabras	3
1.1.2. Autómatas	4
1.2. Construcciones de autómatas	6
1.2.1. Autómatas con función parcial de transición	6
1.2.2. Operaciones de conjuntos	7
1.3. No-determinismo	9
1.3.1. Definición de un NFA	9
1.3.2. Comparación con DFA	11
1.4. Expresiones regulares	13
1.5. Autómatas con transiciones sin lectura	13
1.6. Teorema de Kleene	13
2. Propiedades de lenguajes regulares	14
2.1. Lema de bombeo	14
2.2. Minimización de autómatas	14
2.3. Teorema de Myhill-Nerode	14
2.4. Autómatas en dos direcciones	14
3. Algoritmos para lenguajes regulares	15
3.1. Evaluación de expresiones regulares	15
3.2. Transductores	15
3.3. Análisis léxico	15
3.4. Algoritmo de Knuth-Morris-Prat	15
4. Lenguajes libres de contexto	16
4.1. Gramáticas libres de contexto	16
4.2. Simplificación de gramáticas	16
4.3. Forma normal de Chomsky	16
4.4. Lema de bombeo para lenguajes libres de contexto	16
4.5. Algoritmo CKY	16
5. Algoritmos para lenguajes libres de contexto	17
5.1. Autómatas apiladores	17
5.1.1. Versión normal	17
5.1.2. Versión alternativa	19
5.2. Autómatas apiladores vs gramáticas libres de contexto	21
5.2.1. Desde CFG a PDA	22
5.2.2. Desde PDA a CFG	23
5.3. Parsing: cómputo de First y Follow	25
5.3.1. Prefijos	27
5.3.2. First y Follow	28
5.3.3. Calcular First	29
5.3.4. Calcular Follow	30
5.4. Gramáticas LL	31
5.4.1. Definición Gramáticas LL	32
5.4.2. Caracterización LL	34
5.5. Parsing con gramáticas LL(k)	36
5.5.1. Algunas consideraciones	36
5.5.2. Parsing de LL(k)	39

6. Extracción de información	43
6.1. Extracción	43
6.2. Enumeración de resultados: Autómatas con anotaciones	43

1. Lenguajes regulares

1.1. Palabras y autómatas

1.1.1. Palabras

Definiciones. Consideremos que:

- ♦ Un **alfabeto** Σ es con conjunto finito.
- ♦ Un elemento de Σ lo llamaremos una **letra** o **símbolo**.
- ♦ Una **palabra** o **string** sobre Σ es una secuencia finita de letras en Σ .

Ejemplo 1.1

- $\Sigma = \{a, b, c\}$

- Palabras sobre Σ :

$aaaaabb, bcaabab, a, bbbbbb, \dots$

- ♦ El largo $|w|$ de una palabra w es el número de letras.

$$|w| \stackrel{\text{def}}{=} \# \text{ de letras en } w$$

- ♦ Denotaremos ϵ como la **palabra sin símbolos** de largo 0.

$$|\epsilon| \stackrel{\text{def}}{=} 0$$

- ♦ Denotaremos por Σ^* como el **conjunto de todas las palabras** sobre Σ .

Ejemplo 1.2

Para $\Sigma = \{0, 1\}$:

- $|00011001| = 8$
- $\Sigma^* =$ todas las palabras posibles formadas por 0s y 1s.

Definición. Dados dos palabras $u, v \in \Sigma^*$ tal que $u = a_1 \dots a_n$ y $v = b_1 \dots b_m$:

$$u \cdot v \stackrel{\text{def}}{=} a_1 \dots a_n b_1 \dots b_m$$

Decimos que $u \cdot v$ es la palabra “ **u concatenada con v** ”.

Ejemplo 1.3

Para $\Sigma = \{0, 1, 2, \dots, 9\}$:

- ♦ $0123 \cdot 9938 = 01239938$ y $3493 \cdot \epsilon = 3493$

Propiedades de concatenación. La concatenación cumple:

- ♦ **Asociatividad:** $(u \cdot v) \cdot w = u \cdot (v \cdot w)$
- ♦ **Largo:** $|u \cdot v| = |u| + |v|$
- ♦ ¿Cumple conmutatividad? No. Por ejemplo, si $u = ab$ y $v = bb$, entonces $u \cdot v = abbb \neq bbab = v \cdot u$.

Definición. Sea Σ un alfabeto y $L \subseteq \Sigma^*$. Decimos que L es un **lenguaje** sobre el alfabeto Σ .

Ejemplo 1.4

Sea $\Sigma = \{a, b\}$:

- ♦ $L_0 = \{\epsilon, a, aa, b, aa\}$
- ♦ $L_1 = \{\epsilon, b, bb, bbb, bbbb, \dots\}$
- ♦ $L_2 = \{w \mid \exists u \in L_1, w = a \cdot u\}$
- ♦ $L_3 = \{w \mid \exists u, v \in \Sigma^*, w = u \cdot abba \cdot v\}$
- ♦ $L_4 = \{w \mid \exists u \in \Sigma^*, w = u \cdot u\}$

Un **lenguaje** puede ser visto como una **propiedad** de palabras.

Convenciones. Durante todo este texto:

- ♦ Para **letras** se usarán los símbolos: a, b, c, d, e, \dots
- ♦ Para **palabras** se usarán los símbolos: w, u, v, x, y, z, \dots
- ♦ Para **alfabetos** se usarán los símbolos: Σ, Γ, \dots
- ♦ Para **lenguajes** se usarán los símbolos: L, M, N, \dots
- ♦ Para **números** se usarán los símbolos: i, j, k, l, m, n, \dots

1.1.2. Autómatas

Una autómata **finito** es:

- ♦ Un modelo de computación sencillo, basado en una cantidad **finita** de memoria.
- ♦ Procesa cada **palabra** de principio a fin en **una sola pasada**.
- ♦ Al terminar, el autómata decide si **acepta** o **rechaza** el input.

Usaremos los autómatas finitos para definir **lenguajes**.

Definición. Un autómata finito determinista (DFA) es una tupla:

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$

- ♦ Q es un conjunto finito de **estados**.
- ♦ Σ es el alfabeto del **input**.
- ♦ $\delta : Q \times \Sigma \rightarrow Q$ es la función de **transición**.
- ♦ $q_0 \in Q$ es el **estado inicial**.
- ♦ $F \subseteq Q$ es el conjunto de **estados finales** (o aceptación).

Ejemplo 1.5

$$\diamond Q = \{0, 1, 2\}$$

$$\diamond \Sigma = \{a, b\}$$

$\diamond \delta : Q \times \Sigma \rightarrow Q$ se define como:

$$\delta(0, a) = 1$$

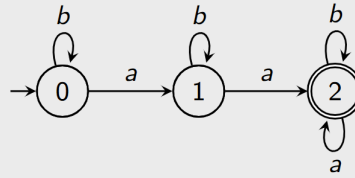
$$\delta(1, a) = 2$$

$$\delta(2, a) = 2$$

$$\delta(q, b) = q \quad \forall q \in \{0, 1, 2\}$$

$$\diamond q_0 = 0$$

$$\diamond F = \{2\}$$

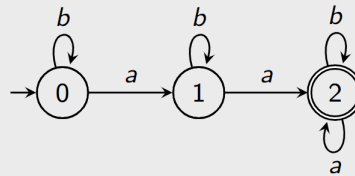


Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA y $w = a_1 a_2 \dots a_n \in \Sigma^*$ un input. Una **ejecución** (o *run*) ρ de \mathcal{A} sobre w es una secuencia:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} p_n$$

donde $p_0 = q_0$ y para todo $i \in \{0, 1, \dots, n-1\}$ se cumple que $\delta(p_i, a_{i+1}) = p_{i+1}$.

Una ejecución ρ de \mathcal{A} sobre w es de **aceptación** si $p_n \in F$.

Ejemplo 1.6

\diamond ¿Cuál es la ejecución de \mathcal{A} sobre $bbab$?

$\rho : 0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 1$. La ejecución **no** es de aceptación ya que no termina en un estado final.

\diamond ¿Cuál es la ejecución de \mathcal{A} sobre $abab$?

$\rho : 0 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{a} 2 \xrightarrow{b} 2$. La ejecución **si** es de aceptación ya que termina en un estado final.

Aceptación. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA y $w \in \Sigma^*$. Decimos que \mathcal{A} **acepta** w si la ejecución de \mathcal{A} sobre w es de aceptación. Al contrario, decimos que \mathcal{A} **rechaza** w si la ejecución de \mathcal{A} sobre w NO es de aceptación.

El **lenguaje aceptado** por \mathcal{A} se define como:

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}$$

Un lenguaje $L \subseteq \Sigma^*$ se dice **regular** si, y sólo si, **existe** un autómata finito determinista \mathcal{A} tal que

$$L = \mathcal{L}(\mathcal{A})$$

1.2. Construcciones de autómatas

Veremos una definición alternativa al autómata visto en la sección anterior.

1.2.1. Autómatas con función parcial de transición

Definición. Un autómata finito determinista con **función parcial de transición** (DFAp) es una tupla:

$$\mathcal{A} = (Q, \Sigma, \gamma, q_0, F)$$

- ♦ Q es un conjunto finito de estados.
- ♦ Σ es el alfabeto del input.
- ♦ $\gamma : Q \times \Sigma \rightarrow Q$ es una **función parcial de transición**.
- ♦ $q_0 \in Q$ es el estado inicial.
- ♦ $F \subseteq Q$ es el conjunto de estados finales (o aceptación).

Ejecución. Sea $w = a_1 a_2 \dots a_n \in \Sigma^*$. De igual manera que un DFA, una **ejecución** (o *run*) ρ de \mathcal{A} sobre w es una secuencia:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_n} p_n$$

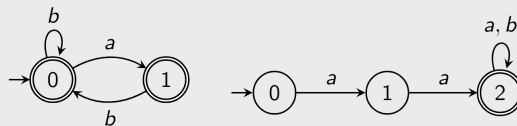
donde $p_0 = q_0$ y para todo $i \in \{0, \dots, n-1\}$ **está definido** $\gamma(p_i, a_{i+1}) = p_{i+1}$.

Una ejecución ρ de \mathcal{A} sobre w es de **aceptación** si $p_n \in F$. Notemos que ahora una palabra puede NO tener una ejecución.

Aceptación. Sea \mathcal{A} un DFAp y $w \in \Sigma^*$. Decimos que \mathcal{A} **acepta** w si **existe una ejecución** de \mathcal{A} sobre w que es de aceptación. También, el **lenguaje aceptado** por \mathcal{A} se define como:

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}$$

Ejemplo 1.7



¿DFA \neq DFAp? Establezcamos una proposición. Para todo autómata \mathcal{A} con función parcial de transición, existe un autómata \mathcal{A}' (con función total de transición) tal que:

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$$

En otras palabras, DFA \equiv DFAp.

Demostración. Sea $\mathcal{A} = (Q, \Sigma, \gamma, q_0, F)$ un autómata con función parcial de transición. Sea q_s un **nuevo estado** tal que $q_s \notin Q$. Construimos el DFA $\mathcal{A}' = (Q \cup \{q_s\}, \Sigma, \delta', q_0, F)$ tal que:

$$\delta'(p, a) = \begin{cases} \gamma(p, a) & \text{si } p \neq q_s \text{ y } (p, a) \in \text{dom}(\gamma) \\ q_s & \text{si no} \end{cases}$$

para todo $p \in Q \cup \{q_s\}$ y $a \in \Sigma$. Queremos demostrar que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Dem. $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$. Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$. Entonces, existe una ejecución de aceptación ρ de \mathcal{A} sobre w :

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_n} p_n$$

donde $p_0 = q_0$, para todo $i \in \{0, \dots, n-1\}$ está definido $\gamma(p_i, a_{i+1}) = p_{i+1}$ y $p_n \in F$. Como $\delta'(p_i, a_{i+1}) = \gamma(p_i, a_{i+1})$ para todo $i \in \{0, \dots, n-1\}$ (por la definición de δ'), entonces ρ es también una ejecución de aceptación de \mathcal{A}' sobre w . Por lo tanto, $w \in \mathcal{L}(\mathcal{A}')$.

Dem. $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$. Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}')$. Entonces, existe una ejecución de aceptación ρ de \mathcal{A}' sobre w :

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_n} p_n$$

donde $p_0 = q_0$, para todo $i \in \{0, \dots, n-1\}$ tenemos $\gamma'(p_i, a_{i+1}) = p_{i+1}$ y $p_n \in F$. Demostraremos que $p_i \neq q_s$ para todo $i \in \{0, \dots, n\}$. Por **contradicción**, suponga que existe i tal que $p_i = q_s$. entonces, tenemos que $p_{i+1} = q_s$. Por **inducción**, podemos demostrar que $p_j = q_s$ para todo $j \geq i$, y así, podemos concluir que $p_n = q_s$, llevándonos a una contradicción. Como $p_i \neq q_s$ para todo $i \in \{0, \dots, n\}$, tenemos que:

$$\delta'(p_i, a_{i+1}) = \gamma(p_i, a_{i+1}) \quad \forall i \in \{0, 1, \dots, n-1\}$$

y entonces ρ es una **ejecución de aceptación** de \mathcal{A} sobre w . Por lo tanto, concluimos que $w \in \mathcal{L}(\mathcal{A})$. ■

Advertencia. Desde ahora, se utilizarán autómatas con funciones **totales** de transición, pero sin pérdida de generalidad, en algunos ejemplos habrán autómatas con funciones **parciales** de transición por simplicidad.

1.2.2. Operaciones de conjuntos

Definiciones. Dado dos lenguajes $L, L' \subseteq \Sigma^*$ se define:

$$\begin{aligned} L^C &= \{w \in \Sigma^* \mid w \notin L\} \\ L \cap L' &= \{w \in \Sigma^* \mid w \in L \wedge w \in L'\} \\ L \cup L' &= \{w \in \Sigma^* \mid w \in L \vee w \in L'\} \end{aligned}$$

Dado dos autómatas \mathcal{A} y \mathcal{A}' :

1. ¿Existe un autómata \mathcal{B} tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})^C$?
2. ¿Existe un autómata \mathcal{B} tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$?
3. ¿Existe un autómata \mathcal{B} tal que $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$?

Construcción de $\mathcal{L}(\mathcal{A})^C$. Dado un autómata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, definimos el autómata:

$$\mathcal{A}^C = (Q, \Sigma, \delta, q_0, Q \setminus F)$$

Teorema 1

Para todo autómata \mathcal{A} , se tiene que $\mathcal{L}(\mathcal{A})^C = \mathcal{L}(\mathcal{A}^C)$.

Producto de autómatas. Suponga que:

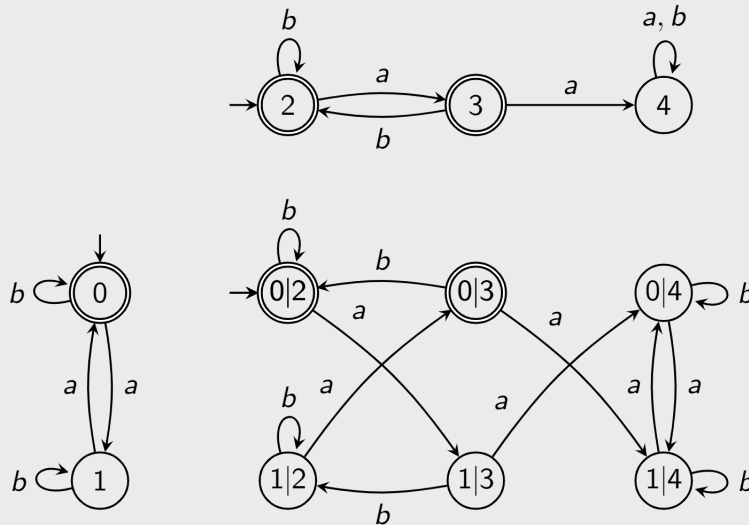
$$\begin{aligned}\mathcal{A} &= (Q, \Sigma, \delta, q_0, F) \\ \mathcal{A}' &= (Q', \Sigma, \delta', q'_0, F')\end{aligned}$$

y considere una palabra $w \in \Sigma^*$. ¿Cómo ejecutamos ambos autómatas sobre w al **mismo tiempo**? La idea es ejecutar \mathcal{A} y \mathcal{A}' **en paralelo**. Así, definimos el **producto** entre \mathcal{A} y \mathcal{A}' como el autómata $\mathcal{A} \times \mathcal{A}' = (Q^\times, \Sigma, \delta^\times, q_0^\times, F^\times)$ tal que:

- ♦ $Q^\times = Q \times Q' = \{(q, q') \mid q \in Q \wedge q' \in Q'\}$
- ♦ $\delta^\times((q, q'), a) = (\delta(q, a), \delta'(q', a))$
- ♦ $q_0^\times = (q_0, q'_0)$
- ♦ $F^\times = F \times F'$

Ejemplo 1.8

Todas las palabras sobre $\{a, b\}$ con una cantidad par de a -letras tal que no hay dos a -letras seguidas.



Teorema 2

Para todo par de autómatas \mathcal{A} y \mathcal{A}' se tiene que

$$\mathcal{L}(\mathcal{A} \times \mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$$

Demostración teorema 2. Solo se demostrará que $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A} \times \mathcal{A}')$, la otra dirección queda propuesta para el lector.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$. Entonces $w \in \mathcal{L}(\mathcal{A})$ y $w \in \mathcal{L}(\mathcal{A}')$. Existen ejecuciones de aceptación ρ y ρ' de \mathcal{A} y \mathcal{A}' sobre w , respectivamente:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n \quad \rho' : p'_0 \xrightarrow{a_1} p'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p'_n$$

- ♦ $p_0 = q_0$ y $p'_0 = q'_0$.
- ♦ $\delta(p_{i-1}, a_i) = p_i$ y $\delta'(p'_{i-1}, a_i) = p_i$ para todo $i \in \{1, \dots, n\}$.
- ♦ $p_n \in F$ y $p'_n \in F'$.

Por definición, tenemos que: $\rho^\times : (p_0, p'_0) \xrightarrow{a_1} (p_1, p'_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (p_n, p'_n)$

- ♦ $(p_0, p'_0) = (q_0, q'_0)$.
- ♦ $(p_i, p'_i) = (\delta(p_{i-1}, a_i), \delta'(p'_{i-1}, a_i)) = \delta^\times((p_{i-1}, p'_{i-1}), a_i) \forall i \in \{1, \dots, n\}$.
- ♦ $(p_n, p'_n) \in F \times F'$.

Por lo tanto, ρ^\times es una ejecución de $\mathcal{A} \times \mathcal{A}'$ sobre w y $w \in \mathcal{L}(\mathcal{A} \times \mathcal{A}')$. ■

Unión de autómatas. Sabemos que

$$\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}') = \left(\mathcal{L}(\mathcal{A})^C \cap \mathcal{L}(\mathcal{A}')^C \right)^C$$

Para calcular el autómata que acepta el lenguaje $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{A}')$:

1. Complementamos \mathcal{A} y \mathcal{A}' .
2. Intersectamos \mathcal{A}^C y $(\mathcal{A}')^C$.
3. Complementamos $\mathcal{A}^C \times (\mathcal{A}')^C$.

1.3. No-determinismo

*“Indeterminism is the concept that events (certain events, or events of certain types) are not caused deterministically (cf. causality) by prior events. It is the opposite of **determinism** and related to chance. It is highly relevant to the philosophical problem of **free will**.”* - Wikipedia.

1.3.1. Definición de un NFA

Definición. Un autómata finito **no-determinista** (NFA) es una estructura:

$$\mathcal{A} = (Q, \Sigma, \Delta, I, F)$$

- ♦ Q es un conjunto finito de estados.
- ♦ Σ es el alfabeto del input.
- ♦ $F \subseteq Q$ es el conjunto de estados finales (o aceptación).
- ♦ $\Delta \subseteq Q \times \Sigma \times Q$ es la **relación de transición**.
- ♦ $I \subseteq Q$ es un **conjunto de estados iniciales**.

Ejemplo 1.9

♦ $Q = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $I = \{0, 1\}$, $F = \{2\}$

♦ $\Delta \subseteq Q \times \Sigma \times Q$ se define como:

$(0, a, 0) \in \Delta$

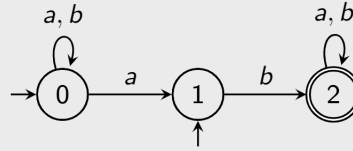
$(0, a, 1) \in \Delta$

$(0, b, 0) \in \Delta$

$(1, b, 2) \in \Delta$

$(2, a, 2) \in \Delta$

$(2, b, 2) \in \Delta$



Ejecución. Sea $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ un NFA y $w = a_1 a_2 \dots a_n \in \Sigma^*$ el input. Una **ejecución** (o *run*) ρ de \mathcal{A} sobre w es una secuencia:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$$

donde $p_0 \in I$ y para todo $i \in \{0, \dots, n-1\}$, se tiene que $(p_i, a_{i+1}, p_{i+1}) \in \Delta$.

Una ejecución ρ de \mathcal{A} sobre w es de **aceptación** si $p_n \in F$.

Aceptación. Decimos que \mathcal{A} **acepta** w si **existe** una ejecución de \mathcal{A} sobre w que es de aceptación. Por otro lado, decimos que \mathcal{A} **rechaza** si **todas** las ejecuciones de \mathcal{A} sobre w NO son de aceptación. Además, el **lenguaje aceptado** por \mathcal{A} se define como:

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ acepta } w\}$$

Interpretación. Las siguientes interpretaciones pueden ayudar a entender mejor un NFA:

1. $\Delta \subseteq Q \times \Sigma \times Q$ es la **relación de transición**.

“(q, a, p) ∈ Δ entonces existe una transición desde q a p al leer a”.

2. $I \subseteq Q$ es un **conjunto de estados iniciales**.

“p ∈ I entonces p es un posible estado inicial del autómata”

1'. $\Delta : Q \times \Sigma \rightarrow 2^Q$ es una **función de transición**.

“q ∈ Δ(p, a) entonces q es un posible estado que puedo llegar desde p al leer a”.

Esta interpretación es más común encontrarla en libros sobre teoría de autómatas.

Además, el **no-determinismo** puede ser visto como:

1. Paralelización infinita, es decir, cada ejecución es un *thread* distinto.

2. “Guessing and Verifying” (adivinar y verificar).

El no-determinismo NO debe ser visto como:

♦ Explicitamente como el **indeterminismo** o “libre albedrío”. Para un input, un NFA siempre produce el mismo resultado.

♦ Comportamiento **aleatorio** del autómata.

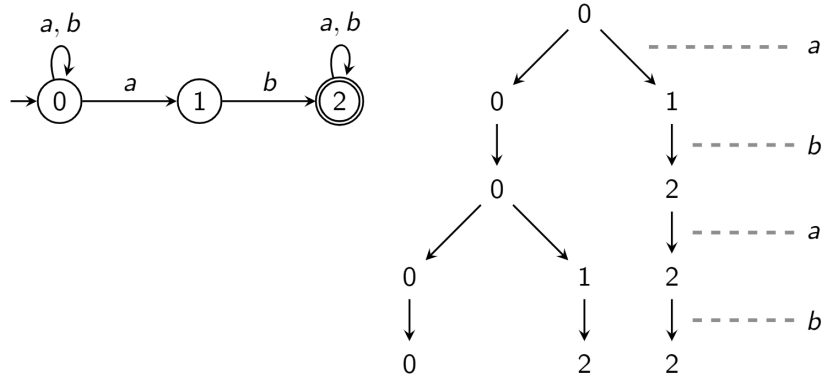


Figura 1: Interpretación del no-determinismo

1.3.2. Comparación con DFA

A continuación, veremos que un autómata finito determinista (DFA) puede almacenar **todas** las ejecuciones de un NFA. A este proceso se le conoce como **determinización**.

Teorema 3

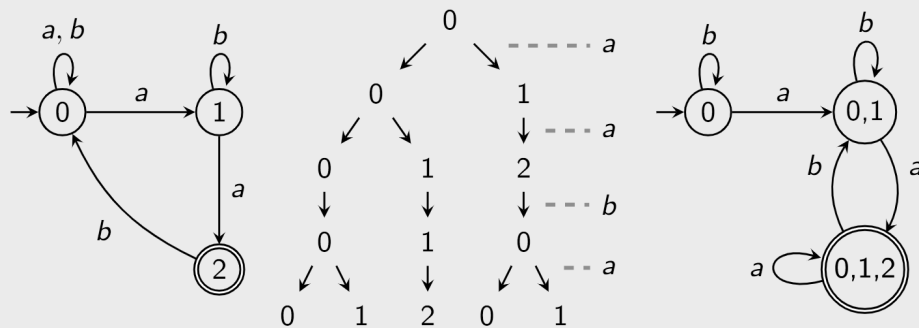
Para todo autómata finito no-determinista \mathcal{A} , existe un autómata determinista \mathcal{A}' tal que

$$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$$

En otras palabras, $DFA \equiv NFA$.

Idea. Primero, pensemos en la idea de determinización: “almacenar en el autómata determinista todos los estados actuales de las ejecuciones en curso (sin repetidos)”.

Ejemplo 1.10



Formalización. Para un autómata no-determinista $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, definimos el autómata determinista (determinización de \mathcal{A}):

$$\mathcal{A}^{\text{det}} = (2^Q, \Sigma, \delta^{\text{det}}, q_0^{\text{det}}, F^{\text{det}})$$

♦ $2^Q = \{S \mid S \subseteq Q\}$ es el conjunto potencia de Q .

♦ $q_0^{\text{det}} = I$.

♦ $\delta^{\text{det}} : 2^Q \times \Sigma \rightarrow 2^Q$ tal que:

$$\delta^{\text{det}}(S, a) = \{q \in Q \mid \exists p \in S. (p, a, q) \in \Delta\}$$

♦ $F^{\text{det}} = \{S \in 2^Q \mid S \cap F \neq \emptyset\}$, es decir, todos los conjuntos que tengan al menos un estado final.

Demostración teorema 3. La determinización puede verse como un **subset construction**. Partamos con $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^{\text{det}})$.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$. Existe una ejecución ρ de \mathcal{A} sobre w :

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$$

donde $p_0 = I$, $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ para todo $i \in \{0, \dots, n-1\}$ y $p_n \in F$.

Como \mathcal{A}^{det} es determinista, entonces existe una ejecución ρ' de \mathcal{A}^{det} sobre w :

$$\rho' : S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} S_n$$

donde $S_0 = I$ y $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1}$ para todo $i \in \{0, \dots, n-1\}$. Luego, queremos demostrar que $p_i \in S_i$ para todo $i \in \{0, \dots, n-1\}$.

Por **inducción** sobre i , tenemos que:

♦ **Caso base:** $p_0 \in S_0$ por definición de \mathcal{A}^{det} .

♦ **Inducción:** Suponemos que $p_i \in S_i$ y demostramos para $i+1$. Como sabemos que:

- $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1} = \{q \in Q \mid \exists p \in S_i. (p, a, q) \in \Delta\}$ y
- $(p_i, a_{i+1}, p_{i+1}) \in \Delta$

Entonces $p_{i+1} \in S_{i+1}$, ya que, si estamos en p_i leyendo a_{i+1} , la transición nos dice que pasaremos a al estado p_{i+1} que pertenece a S_{i+1} por la hipótesis de inducción.

Luego, como $p_n \in S_n$, entonces $S_n \cap F \neq \emptyset$ y así $S_n \in F^{\text{det}}$. Por lo tanto, $w \in \mathcal{L}(\mathcal{A}^{\text{det}})$.

Ahora, demostramos la otra dirección: $\mathcal{L}(\mathcal{A}^{\text{det}}) \subseteq \mathcal{L}(\mathcal{A})$.

Sea $w = a_1 \dots a_n \in \mathcal{L}(\mathcal{A}^{\text{det}})$. Existe una ejecución ρ de \mathcal{A}^{det} sobre w :

$$\rho : S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} S_n$$

donde $S_0 = I$, $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1}$ para todo $i \in \{0, \dots, n-1\}$ y $S_n \in F^{\text{det}}$, con $S_n \cap F \neq \emptyset$. Buscamos demostrar entonces que para todo $i \leq n$ y para todo $p \in S_i$, existe una ejecución:

$$\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i = p$$

tal que:

1. $p_0 \in I$.
2. $(p_j, a_{j+1}, p_{j+1}) \in \Delta$ para todo $j \in \{0, \dots, i-1\}$.

Por **inducción** sobre i , tenemos que:

♦ **Caso base:** Si $p \in S_0 = I$, entonces la ejecución $\rho : p$ cumple 1. y 2.

♦ **Inducción:** Supongamos que se cumple para todo $p \in S_i$. Sea $q \in S_{i+1}$. Como $\delta^{\text{det}}(S_i, a_{i+1}) = S_{i+1} = \{q \in Q \mid \exists p \in S_i. (p, a, q) \in \Delta\}$ y $q \in S_{i+1}$, entonces existe $p \in S_i$ tal que $(p, a_{i+1}, q) \in \Delta$.

Por hipótesis de inducción, existe $\rho : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i = p$ que satisface 1. y 2.

Por lo tanto, $\rho' : p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_i} p_i \xrightarrow{a_{i+1}} q$ también satisface 1. y 2.

Como lo anterior queda demostrado y como $S_n \cap F \neq \emptyset$, para $p \in S_n \cap F$ existe una ejecución de aceptación de \mathcal{A} sobre w . Por lo tanto, $w \in \mathcal{L}(\mathcal{A})$. ■

- 1.4. Expresiones regulares
- 1.5. Autómatas con transiciones sin lectura
- 1.6. Teorema de Kleene

2. Propiedades de lenguajes regulares

- 2.1. Lema de bombeo
- 2.2. Minimización de autómatas
- 2.3. Teorema de Myhill-Nerode
- 2.4. Autómatas en dos direcciones

3. Algoritmos para lenguajes regulares

- 3.1. Evaluación de expresiones regulares
- 3.2. Transductores
- 3.3. Análisis léxico
- 3.4. Algoritmo de Knuth-Morris-Prat

4. Lenguajes libres de contexto

4.1. Gramáticas libres de contexto

4.2. Simplificación de gramáticas

4.3. Forma normal de Chomsky

4.4. Lema de bombeo para lenguajes libres de contexto

4.5. Algoritmo CKY

5. Algoritmos para lenguajes libres de contexto

5.1. Autómatas apiladores

5.1.1. Versión normal

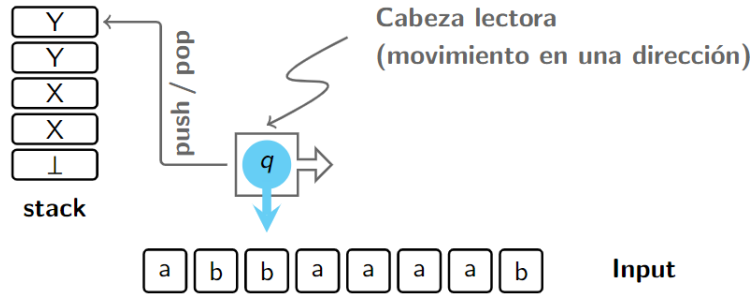


Figura 2: Idea de un autómata apilador

Definición. Un autómata apilador (*PushDown Automata*, PDA) es una estructura:

$$\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$$

- ♦ Q es un conjunto finito de **estados**.
- ♦ Σ es el alfabeto del **input**.
- ♦ $q_0 \in Q$ es el estado **inicial**.
- ♦ F es el conjunto de estados **finales**.
- ♦ Γ es el alfabeto de **stack**.
- ♦ $\perp \in \Gamma$ es el símbolo **inicial del stack** (fondo).
- ♦ $\Delta \subseteq (Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$ es una relación finita de transición.

Intuitivamente, la transición:

$$\left((p, a, A), (q, B_1 B_2 \cdots B_k) \right) \in \Delta$$

si el autómata apilador está:

- ♦ en el estado p , leyendo a , y en el tope del stack hay una A ,
- entonces:
- ♦ cambia al estado q , y modifico el tope A por $B_1 B_2 \cdots B_k$.

Intuitivamente, la transición **en vacío**:

$$\left((p, \epsilon, A), (q, B_1 B_2 \cdots B_k) \right) \in \Delta$$

si el autómata apilador está:

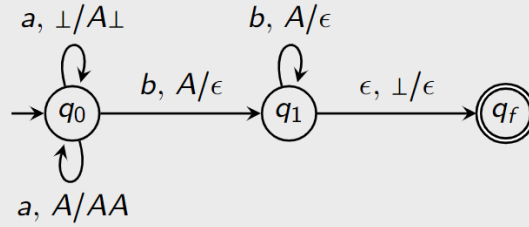
- ♦ en el estado p , *sin lectura de una letra*, y en el tope del stack hay una A ,
- entonces:
- ♦ cambia al estado q , y modifico el tope A por $B_1 B_2 \cdots B_k$.

Ejemplo 5.1

$$\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, \{q_f\})$$

♦ $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \perp\}$ y Δ :

$$\begin{array}{ll} (q_0, a, \perp, q_0, A \perp) & q_0 \perp \xrightarrow{a} q_0 A \perp \\ (q_0, a, A, q_0, AA) & q_0 A \xrightarrow{a} q_0 AA \\ (q_0, b, A, q_1, \epsilon) & q_0 A \xrightarrow{b} q_1 \\ (q_1, b, A, q_1, \epsilon) & q_1 A \xrightarrow{b} q_1 \\ (q_1, \epsilon, \perp, q_f, \epsilon) & q_1 \perp \xrightarrow{\epsilon} q_f \end{array}$$



Notación. Dada una palabra $A_1 A_2 \dots A_k \in \Gamma^+$ decimos que:

- ♦ $A_1 A_2 \dots A_k$ es un **stack** (contenido),
- ♦ A_1 es el **tope** del stack y
- ♦ $A_2 \dots A_k$ es la **cola** del stack.

Definición. Una **configuración** de \mathcal{P} es una tupla $(q \cdot \gamma, w) \in (Q \cdot \Gamma^*, \Sigma^*)$ tal que:

- ♦ q es el estado actual.
- ♦ γ es el contenido del stack.
- ♦ w es el contenido del input.

Decimos que una configuración:

$$(q \cdot \gamma, w) \in (Q \cdot \Gamma^*, \Sigma^*)$$

- ♦ es **inicial** si $q \cdot \gamma = q_0 \cdot \perp$.
- ♦ es **final** si $q \cdot \gamma = q_f \cdot \epsilon$ con $q_f \in F$ y $w = \epsilon$.

Definición. Se define la relación $\vdash_{\mathcal{P}}$ de **siguiente-paso** entre configuraciones de \mathcal{P} :

$$(q_1 \cdot \gamma_1, w_1) \vdash_{\mathcal{P}} (q_2 \cdot \gamma_2, w_2)$$

si, y sólo si, existe una transición $(q_1, a, A, q_2, \alpha) \in \Delta$ y $\gamma \in \Gamma^*$ tal que:

- ♦ $w_1 = a \cdot w_2$

♦ $\gamma_1 = A \cdot \gamma$

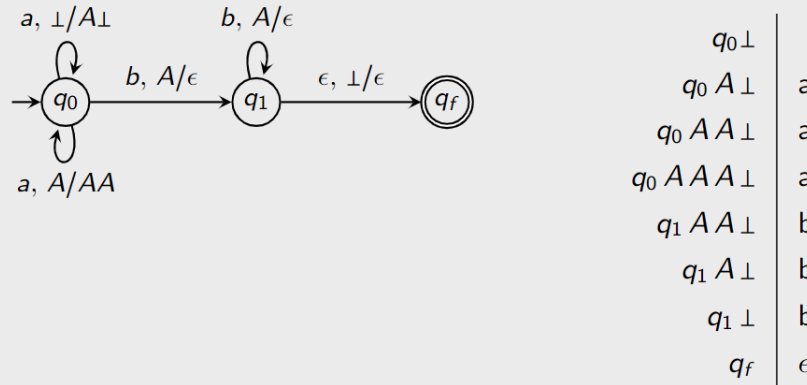
♦ $\gamma_2 = \alpha \cdot \gamma$

Se define $\vdash_{\mathcal{P}}^*$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{P}}$. En otras palabras:

$(q_1 \gamma_1, w_1) \vdash_{\mathcal{P}}^* (q_2 \gamma_2, w_2)$ si uno puede ir de $(q_1 \gamma_1, w_1)$ a $(q_2 \gamma_2, w_2)$ en 0 o más pasos.

Ejemplo 5.2

Para la palabra $w = aaabbb$, tenemos la ejecución:



Definiciones. \mathcal{P} **acepta** w si, y sólo si, $(q_0 \perp, w) \vdash_{\mathcal{P}}^* (q_f, \epsilon)$ para algún $q_f \in F$.

El **lenguaje aceptado** por \mathcal{P} se define como:

$$\mathcal{L}(\mathcal{P}) = \{w \in \Sigma^* \mid \mathcal{P} \text{ acepta } w\}$$

Ejemplo 5.3

El lenguaje aceptado por el PDA utilizado en los ejemplos anteriores es $\mathcal{L}(\mathcal{P}) = \{a^n b^n \mid n \geq 0\}$.

5.1.2. Versión alternativa

Esta definición de autómatas apilador es poco común pero trae algunas ventajas:

- ♦ Es un modelo que ayuda a entender mejor los algoritmos de evaluación para gramáticas.
- ♦ Es un modelo menos estándar pero mucho más sencillo.
- ♦ Al profe Cristian le gustó y lo encontró interesante.

Definición. Un **PDA alternativo** es una estructura:

$$\mathcal{D} = (Q, \Sigma, \Delta, q_0, F)$$

- ♦ Q es un conjunto finito de **estados**.

- ♦ Σ es el alfabeto del **input**.
- ♦ $q_0 \in Q$ es el estado **inicial**.
- ♦ F es el conjunto de estados **finales**.
- ♦ $\Delta \subseteq Q^+ \times (\Sigma \cup \{\epsilon\}) \times Q^*$ es una **relación finita de transición**.

Intuitivamente, la transición:

$$(A_1 \dots A_i, a, B_1 \dots B_j) \in \Delta$$

si el autómata apilador tiene:

- ♦ $A_1 \dots A_i$ en el tope del stack y leyendo a ,

entonces:

- ♦ cambia el tope $A_1 \dots A_i$ por $B_1 \dots B_j$.

En este tipo de autómata apilador, **no hay diferencia** entre estados y alfabeto del stack.

Definición. Una **configuración** de \mathcal{D} es una tupla

$$(q_1 \dots q_k, w) \in (Q^+, \Sigma^*)$$

tal que:

- ♦ $q_1 \dots q_k$ es el contenido del stack con q_1 el tope del stack.
- ♦ w es el contenido del input.

Decimos que una configuración:

- ♦ (q_0, w) es **inicial**.
- ♦ (Q_f, ϵ) es **final** si $q_f \in F$.

Definición. Se define la relación $\vdash_{\mathcal{D}}$ de **siguiente-paso** entre configuraciones de \mathcal{D} :

$$(\gamma_1, w_1) \vdash_{\mathcal{D}} (\gamma_2, w_2)$$

si, y sólo si, existe una transición $(\alpha, a, \beta) \in \Delta$ y $\gamma \in \Gamma^*$ tal que:

- ♦ $w_1 = a \cdot w_2$
- ♦ $\gamma_1 = \alpha \cdot \gamma$
- ♦ $\gamma_2 = \beta \cdot \gamma$

Se define $\vdash_{\mathcal{D}}^*$ como la clausura **refleja** y **transitiva** de $\vdash_{\mathcal{D}}$.

Definiciones. \mathcal{D} **acepta** w si, y sólo si, $(q_0, w) \vdash_{\mathcal{D}}^* (q_f, \epsilon)$ para algún $q_f \in F$. Además, el **lenguaje aceptado** por \mathcal{D} se define como:

$$\mathcal{L}(\mathcal{D}) = \{w \in \Sigma^* \mid \mathcal{D} \text{ acepta } w\}$$

Ejemplo 5.4

$$\mathcal{D} = (Q, \{a, b\}, \Delta, q_0, F)$$

♦ $Q = \{\perp, q_0, q_1, q_f\}$ y Δ :

$(\perp, a, q_0 \perp)$	$\perp \xrightarrow{a} q_0 \perp$	\perp	
$(q_0, a, q_0 q_0)$	$q_0 \xrightarrow{a} q_0 q_0$	$q_0 \perp$	a
(q_0, b, q_1)	$q_0 \xrightarrow{a} q_1$	$q_0 q_0 \perp$	a
$(q_1 q_0, b, q_1)$	$q_1 q_0 \xrightarrow{b} q_1$	$q_0 q_0 q_0 \perp$	a
$(q_1 \perp, \epsilon, q_f)$	$q_1 \perp \xrightarrow{b} q_f$	$q_1 q_0 q_0 \perp$	b
		$q_1 q_0 \perp$	b
		$q_1 \perp$	b
		q_f	ϵ

$$\mathcal{L}(\mathcal{D}) = \{a^n b^n \mid n \geq 1\}$$

Teorema 4

Para todo autómata apilador \mathcal{P} existe un autómata apilador alternativo \mathcal{D} , y viceversa, tal que:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{D})$$

El teorema anterior nos dice que podemos usar ambos modelos de manera **equivalente**.

5.2. Autómatas apiladores vs gramáticas libres de contexto

¿En qué se parecen CFG a PDA?

Gramáticas libre de contexto		Autómatas apiladores
■ Terminales (Σ)		■ Alfabeto input (Σ)
■ Variables (V)		■ Alfabeto stack (Γ)
■ Producciones		■ Transiciones
■ $A \rightarrow \gamma$		■ $pA \xrightarrow{\zeta} q\gamma$
■ $A \rightarrow a$		■ $pA \xrightarrow{\zeta} q$
■ Derivaciones		■ Ejecuciones
■ Árbol de derivación		■ Stack

Figura 3: Gramáticas vs Autómatas apiladores

Teorema 5

Todo **lenguaje libre de contexto** puede ser descrito equivalentemente por:

- ♦ Una gramática libre de contexto (**CFG**).
- ♦ Un autómata apilador (**PDA**).

5.2.1. Desde CFG a PDA

Partimos enunciado un teorema:

Teorema 6

Para toda gramática libre de contexto \mathcal{G} , existe un **autómata apilador alternativo** \mathcal{D} , tal que:

$$\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{D})$$

Construcción \mathcal{D} desde \mathcal{G} . Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG. Construimos un PDA alternativo \mathcal{D} que acepta $\mathcal{L}(\mathcal{G})$:

$$\mathcal{D} = (V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \Delta, q_0, \{q_f\})$$

La relación de transición Δ se define como:

$$\begin{aligned} \Delta = & \{(q_0, \epsilon, S \cdot q_f)\} & \cup \\ & \{(X, \epsilon, \gamma) \mid X \rightarrow \gamma \in P\} & \cup \text{ (Expandir)} \\ & \{(a, a, \epsilon) \mid a \in \Sigma\} & \text{ (Reducir)} \end{aligned}$$

Demostración $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{D})$. Debemos demostrar dos direcciones: $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{D})$ y $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{G})$.

Demostración $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{D})$. Para cada $w \in \mathcal{L}(\mathcal{G})$ debemos encontrar una ejecución de aceptación de \mathcal{D} sobre w . ¿Cómo encontramos esta ejecución? La idea es que para cada árbol de derivación \mathcal{T} de \mathcal{G} sobre w , construimos una ejecución de \mathcal{D} sobre w que recorre el árbol \mathcal{T} **en profundidad** (DFS). Por tanto, debemos usar **inducción** sobre la altura del árbol \mathcal{T} .

Hipótesis de inducción. Para todo árbol de derivación \mathcal{T} de \mathcal{G} con **altura** h tal que:

♦ la raíz de \mathcal{T} es X , y

♦ \mathcal{T} produce la palabra w

entonces $(X \cdot \gamma, w) \vdash_{\mathcal{D}}^* (\gamma, \epsilon)$ para todo $\gamma \in Q^+$.

Caso base: $h = 1$. Si \mathcal{T} tiene altura 1, entonces:

♦ \mathcal{T} produce la palabra $w = a$ para algún $a \in \Sigma$ y

♦ \mathcal{T} consiste de un nodo X y un hijo a con $X \rightarrow a$.

Entonces para todo $\gamma \in Q^+$:

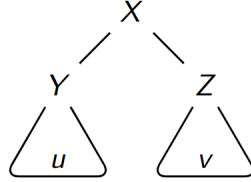
$$(X \cdot \gamma, a) \vdash_{\mathcal{D}} (a \cdot \gamma, a) \vdash_{\mathcal{D}} (\gamma, \epsilon)$$

es una ejecución de \mathcal{D} sobre a .

Caso inductivo: $h = n$. Suponemos que el árbol de derivación \mathcal{T} de \mathcal{G} tiene **altura** n tal que:

- ♦ la raíz de \mathcal{T} es X , y
- ♦ \mathcal{T} produce la palabra w .

Sin pérdida de generalidad, suponga que \mathcal{T} es de la forma:



donde $w = u \cdot v$ y $X \rightarrow YZ$. Por HI, se tiene que para todo $\gamma_1, \gamma_2 \in Q^+$:

$$\begin{aligned} (Y \cdot \gamma_1, u) &\vdash_{\mathcal{D}}^* (\gamma_1, \epsilon) \\ (Z \cdot \gamma_2, v) &\vdash_{\mathcal{D}}^* (\gamma_2, \epsilon) \end{aligned}$$

Para $\gamma \in Q^+$ **construimos** la siguiente ejecución de \mathcal{D} sobre $w = uv$:

$$(X \cdot \gamma, uv) \vdash_{\mathcal{D}} (YZ \cdot \gamma, uv) \vdash_{\mathcal{D}}^* (Z \cdot \gamma, v) \vdash_{\mathcal{D}}^* (\gamma, \epsilon)$$

■

La demostración de $\mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{G})$ se deja como ejercicio propuesto al lector.

5.2.2. Desde PDA a CFG

Partimos enunciando el siguiente teorema:

Teorema 7

Para todo autómata apilador \mathcal{P} , existe una gramática libre de contexto \mathcal{G} tal que:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{G})$$

Demostración $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{G})$. Sea $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ un PDA (normal). Los pasos a seguir son:

1. Convertir \mathcal{P} a un PDA \mathcal{P}' con **UN solo estado**.
2. Convertir \mathcal{P}' a una gramática libre de contexto \mathcal{G} .

Paso 1. Sea $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, F)$ un PDA. Podemos analizar:

- ♦ ¿Por qué NO necesitamos la información de los estados?
- ♦ ¿Cómo guardamos la información de los estados en el stack?

Esto conlleva a la siguiente pregunta: *Si el PDA está en el estado p y en el tope del stack hay una A , ¿a cuál estado llegaré al remover A del stack?*

La solución a esta pregunta es que podemos **adivinar** (no-determinismo) el estado que vamos a llegar cuando removamos A del stack.

Sin pérdida de generalidad, podemos asumir que

1. Todas las transiciones son de la forma:

$$qA \xrightarrow{c} pB_1B_2 \quad \text{o} \quad qA \xrightarrow{c} p\epsilon$$

con $c \in (\Sigma \cup \{\epsilon\})$.

2. Existe $q_f \in Q$ tal que si $w \in \mathcal{L}(\mathcal{P})$ entonces:

$$(q_0 \perp, w) \vdash_{\mathcal{D}}^* (q_f, \epsilon)$$

Estos dos puntos nos aseguran que siempre llegamos al **mismo estado** q_f . Luego, construimos el autómata apilador \mathcal{P}' con **un solo estado**:

$$\mathcal{P}' = (\{q\}, \Sigma, \Gamma', \Delta', \{q\}, \perp', \{q\})$$

♦ $\Gamma' = Q \times \Gamma \times Q$.

“ $(p, A, q) \in \Gamma'$ si desde p leyendo A en el tope del stack llegamos a q al hacer pop de A ”.

♦ $\perp' = (q_0, \perp, q_f)$.

“El autómata parte en q_0 y al hacer pop de \perp llegará a q_f ”.

♦ Si $pA \xrightarrow{c} p'B_1B_2 \in \Delta$ con $c \in (\Sigma \cup \{\epsilon\})$, entonces **para todo** $p_1, p_2 \in Q$:

$$q(p, A, p_2) \xrightarrow{c} q(p', B_1, p_1) \quad (p_1, B_2, p_2) \in \Delta'$$

♦ Si $pA \xrightarrow{c} p' \in \Delta$ con $c \in (\Sigma \cup \{\epsilon\})$, entonces:

$$q(p, A, p') \xrightarrow{c} q \in \Delta'$$

Hipótesis de inducción (en el número de pasos n). Para todo $p, p' \in Q$, $A \in \Gamma$ y $w \in \Sigma^*$ se cumple que:

$$(pA, w) \vdash_{\mathcal{P}}^n (p', \epsilon) \quad \text{si, y solo si,} \quad (q(p, A, p'), w) \vdash_{\mathcal{P}'}^n (q, \epsilon)$$

donde $\vdash_{\mathcal{P}}^n$ es la relación de **siguiente-paso** de \mathcal{P} n -veces.

Si demostramos esta hipótesis, habremos demostrado que $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$. ¿Por qué?

Caso base: $n = 1$. Para todo $p, p' \in Q$, y $A \in \Gamma$ se cumple que:

$$(pA, c) \vdash_{\mathcal{P}} (p', \epsilon) \quad \text{si, y solo si,} \quad (q(p, A, p'), c) \vdash_{\mathcal{P}'} (q, \epsilon)$$

para todo $c \in (\Sigma \cup \{\epsilon\})$.

Caso inductivo. **Sin pérdida de generalidad**, suponga que $pA \xrightarrow{a} p_1A_1A_2$ y $w = auv$, entonces

$$(pA, \underbrace{auv}_w) \vdash_{\mathcal{P}}^n (p', \epsilon) \quad \text{ssi} \quad (pA, auv) \vdash_{\mathcal{P}} (p_1A_1A_2, uv) \vdash_{\mathcal{P}}^i (p_2A_2, v) \vdash_{\mathcal{P}}^j (p', \epsilon)$$

$$\text{ssi} \quad (p_1A_1, u) \vdash_{\mathcal{P}}^i (p_2, \epsilon) \quad \text{y} \quad (p_2A_2, v) \vdash_{\mathcal{P}}^j (p', \epsilon)$$

$$\text{ssi} \quad (q(p_1, A_1, p_2), u) \vdash_{\mathcal{P}'}^i (q, \epsilon) \quad \text{y} \quad (q(p_2, A_2, p'), v) \vdash_{\mathcal{P}'}^j (q, \epsilon)$$

$$\text{ssi} \quad (q(p, A, p'), auv) \vdash_{\mathcal{P}} (q(p_1, A_1, p_2)(p_2, A_2, q)), uv) \vdash_{\mathcal{P}}^{i+j} (q, \epsilon)$$

■

Paso 2. Sea $\mathcal{P} = (\{q\}, \Sigma, \Gamma, \Delta, q, \perp, \{q\})$ un PDA con **UN solo estado**. Construimos la gramática:

$$\mathcal{G} = (V, \Sigma, P, \perp)$$

♦ $V = \Gamma$.

♦ Si $qA \xrightarrow{\epsilon} q\alpha \in \Delta$ entonces $A \rightarrow \alpha \in P$

♦ Si $qA \xrightarrow{a} q\alpha \in \Delta$ entonces $A \rightarrow a\alpha \in P$

La demostración de este paso queda como ejercicio propuesto al lector.

5.3. Parsing: cómputo de First y Follow

Recordatorio. La **sintaxis** de un lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado. Por otro lado, la **semántica** de un lenguaje define el significado de un programa correcto según la sintaxis.

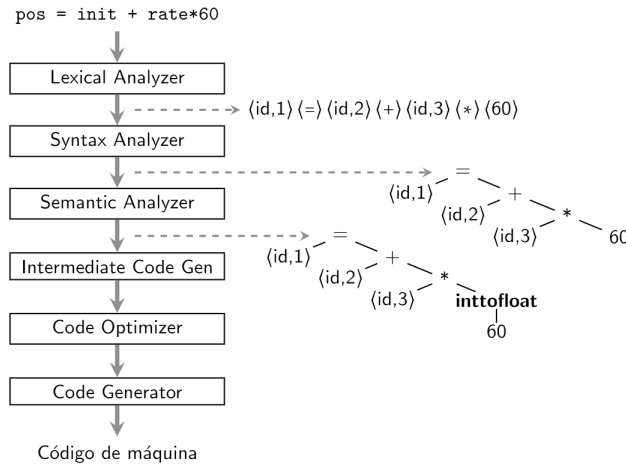


Figura 4: La estructura de un compilador

Lo que se busca es un proceso de **verificación de sintaxis** de un programa, y que entregue la estructura del mismo (árbol de parsing). Consta de tres etapas:

1. Análisis léxico (**Lexer**).
2. Análisis sintático (**Parser**).
3. Análisis semántico.

En una sección anterior vimos el **Lexer**. Ahora, veremos como hacer el **Parser**.

Informalmente: “*dado una secuencia de tokens w' y una gramática \mathcal{G} , construir un árbol de derivación (parsing) de \mathcal{G} para w* ”.

Con el **árbol de derivación** habremos verificado la sintaxis y obtenido la estructura.

Ejemplo 5.5: Parsing de gramática

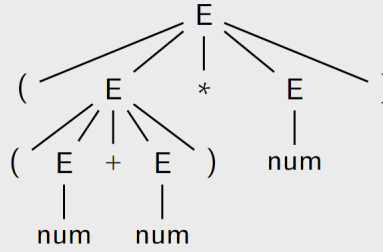
$$E \rightarrow (E + E) \mid (E * E) \mid \text{num}$$

Para un input $w = ((43 + 56) * 27)$:

- ♦ Convertimos w en una secuencia de **tokens**:

$$w' = ((\text{num} + \text{num}) * \text{num})$$

- ♦ Construimos un árbol de **parsing** para w' :



Problema de parsing. Dado una palabra w y dado una gramática \mathcal{G} , generar un árbol de parsing \mathcal{T} de \mathcal{G} para w . ¿Ya sabemos resolver este problema? El algoritmo CKY nos permite hacer esto, pero:

- ♦ es impracticable para grandes inputs.
- ♦ múltiples pasadas sobre el input.

Deseamos hacer parsing en **tiempo lineal** en el tamaño del input. ¿Quién nos puede rescatar ante tal problema? Efectivamente, los autómatas apiladores.

Recordemos que, para una gramática $\mathcal{G} = (V, \Sigma, P, S)$ **podemos construir un PDA alternativo** \mathcal{D} que acepta $\mathcal{L}(\mathcal{G})$:

$$\mathcal{D} = (V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \Delta, q_0, \{q_f\})$$

La relación de transición Δ se define como:

$$\begin{aligned} \Delta = & \{(q_0, \epsilon, S \cdot q_f)\} && \cup \\ & \{(X, \epsilon, \gamma) \mid X \rightarrow \gamma \in P\} && \cup \quad \textbf{(Expandir)} \\ & \{(a, a, \epsilon) \mid a \in \Sigma\} && \textbf{(Reducir)} \end{aligned}$$

Con esto, nos encontramos con otro **problema**: hay muchas alternativas para **expandir**. ¿Cómo elegir entonces la siguiente producción para expandir? Por ejemplo, si tenemos la regla $X \rightarrow \alpha \mid \beta$, ¿cómo elegir entre α o β ?

Queremos elegir la **próxima producción** $X \rightarrow \gamma$ de tal manera que, si existe una derivación para el input, entonces $X \rightarrow \gamma$ **es parte de esa derivación**:

$$\text{si } S \xrightarrow{\text{lm}}^* uX\gamma' \xrightarrow{\text{lm}}^* uv, \text{ entonces } \gamma\gamma' \xrightarrow{\text{lm}}^* v$$

Necesitamos **mirar las siguientes letras en** v y ver si pueden ser producidas por α o β . Para esto, ocuparemos los conceptos de **first** y **follow**.

5.3.1. Prefijos

Definición. Sea Σ un alfabeto finito. Para un $k \geq 0$, se define

$$\begin{aligned}\Sigma^{\leq k} &= \bigcup_{i=0}^k \Sigma^i \\ \Sigma_{\#}^{\leq k} &= \Sigma^{\leq k} \cup (\Sigma^{\leq k-1} \cdot \{\#\})\end{aligned}$$

Ejemplo 5.6

Para $\Sigma = \{a, b\}$:

- ♦ $\Sigma^{\leq 2} = \{\epsilon, a, b, aa, ab, ba, bb\}$
- ♦ $\Sigma_{\#}^{\leq 2} = \{\epsilon, a, b, aa, ab, ba, bb\} \cup \{\#, a\#, b\#\}$

El símbolo $\#$ representará un EOF (End Of File), marcando el **fin de una palabra**.

Definición. Para una palabra $w = a_1 a_2 \dots a_n \in \Sigma^*$ se define el k -**prefijo** de w como:

$$w|_k = \begin{cases} a_1 \dots a_n & \text{si } n \leq k \\ a_1 \dots a_k & \text{si } k < n \end{cases}$$

Definimos la k -**concatenación** \odot_k entre strings $u, v \in \Sigma$ como:

$$u \odot_k v = (u \cdot v)|_k$$

Ejemplo 5.7

Sea $\Sigma = \{a, b\}$, entonces:

- ♦ $(abaa)|_2 = ab \quad (ab)|_2 = ab \quad (a)|_2 = a \quad (\epsilon)|_2 = \epsilon$
- ♦ $a \odot_2 baa = (abaa)|_2 = ab$
- ♦ $bba \odot_2 a = (bbaa)|_2 = bb$
- ♦ $b \odot_2 \epsilon = (b)|_2 = b$

Extendemos estas operaciones para lenguajes $L, L_1, L_2 \subseteq \Sigma^*$ como:

$$\begin{aligned}L|_k &= \{w|_k \mid w \in L\} \\ L_1 \odot_k L_2 &= \{w_1 \odot_k w_2 \mid w_1 \in L_1 \text{ y } w_2 \in L_2\}\end{aligned}$$

Ejemplo 5.8

- ♦ $((ab)^*)|_3 = \{\epsilon, ab, aba\}$
- ♦ $(a)^* \odot_3 (ab)^* = \{\epsilon, a, aa, aaa, ab, aba, aab\}$

Podemos decir que los operadores $|_k$ y \odot_k “miran” hasta un prefijo k .

Propiedades. Para todo $k \geq 1$ y $L_1, L_2, L_3 \subseteq \Sigma^*$:

1. $L_1 \odot_k (L_2 \odot_k L_3) = (L_1 \odot_k L_2) \odot_k L_3$
2. $L_1 \odot_k \{\epsilon\} = \{\epsilon\} \odot_k L_1 = L_1|_k$
3. $(L_1 L_2)|_k = L_1|_k \odot_k L_2|_k$
4. $L_1 \odot_k (L_2 \cup L_3) = (L_1 \odot_k L_2) \cup (L_1 \odot_k L_3)$

La demostración de estas propiedades queda como ejercicio propuesto al lector.

5.3.2. First y Follow

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Definición. Se define la función $\text{first}_k : (V \cup \Sigma)^* \rightarrow 2^{\Sigma^{\leq k}}$ tal que, para $\gamma \in (V \cup \Sigma)^*$:

$$\text{first}_k(\gamma) = \{u|_k \mid \gamma \xRightarrow{*} u\}$$

Ejemplo 5.9

$$E \rightarrow (E + E) \mid (E * E) \mid n$$

- ♦ $\text{first}_1(E) = \{(\, n\}$
- ♦ $\text{first}_2(E) = \{n, (n, (\}$
- ♦ $\text{first}_3(E) = \{n, (n+, (n*, ((n, (((\}$

Definición. Se define la función $\text{follow}_k : V \rightarrow 2^{\Sigma^{\leq k}_{\#}}$ como:

$$\text{follow}_k(X) = \{w \mid S \xRightarrow{*} \alpha X \beta \text{ y } w \in \text{first}_k(\beta\#)\}$$

Ejemplo 5.10

$$E \rightarrow (E + E) \mid (E * E) \mid n$$

- ♦ $\text{follow}_1(E) = \{\#, +, *,)\}$
- ♦ $\text{follow}_2(E) = \{\#,)\#,),)+,)*, +(, *(, +n, *n\}$

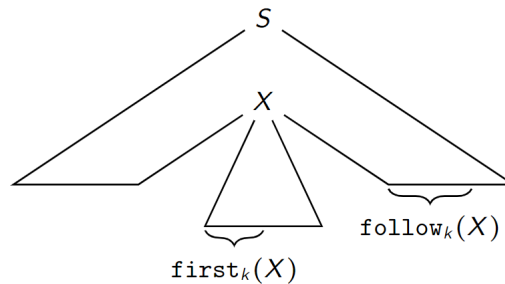


Figura 5: Representación de first y follow

5.3.3. Calcular First

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Proposición. Para $X_1, \dots, X_n \in (V \cup \Sigma)$:

$$\mathbf{first}_k(X_1 \dots X_n) = \mathbf{first}_k(X_1) \odot_k \dots \odot_k \mathbf{first}_k(X_n)$$

Demostración. Defina $\mathcal{L}(X) = \{w \mid X \xRightarrow{*} w\}$ y $\mathcal{L}(\gamma) = \{w \mid \gamma \xRightarrow{*} w\}$. Notar que $\mathbf{first}_k(\gamma) = \mathcal{L}(\gamma)|_k$, por lo tanto, tenemos que

$$\begin{aligned} \mathbf{first}_k(X_1 \dots X_n) &= \mathcal{L}(X_1 \dots X_n)|_k \\ &= (\mathcal{L}(X_1) \cdot \mathcal{L}(X_2) \cdot \dots \cdot \mathcal{L}(X_n))|_k \\ &= \mathcal{L}(X_1)_k \odot_k \mathcal{L}(X_2)_k \odot_k \dots \odot_k \mathcal{L}(X_n)_k \\ &= \mathbf{first}_k(X_1) \odot_k \dots \odot_k \mathbf{first}_k(X_n) \end{aligned}$$

■

En particular, tenemos que:

$$\mathbf{first}_k(X) = \bigcup_{X \rightarrow X_1 \dots X_n \in P} \mathbf{first}_k(X_1) \odot_k \dots \odot_k \mathbf{first}_k(X_n)$$

Definimos el siguiente **programa recursivo** para todo $X \in (V \cup \Sigma)$:

$$\begin{aligned} \mathbf{first}_k^0(X) &:= \bigcup_{X \rightarrow w \in P} w|_k \\ \mathbf{first}_k^i(X) &:= \bigcup_{X \rightarrow X_1 \dots X_n \in P} \mathbf{first}_k^{i-1}(X_1) \odot_k \dots \odot_k \mathbf{first}_k^{i-1}(X_n) \end{aligned}$$

Es fácil ver que:

- ♦ $\mathbf{first}_k^{i-1}(X) \subseteq \mathbf{first}_k^i(X)$ para todo $i > 1$.
- ♦ Como $\mathbf{first}_k(X) \subseteq \Sigma^{\leq k}$, entonces para algún $i \leq k \cdot |\Sigma|^k \cdot |V|$ tendremos:

$$\mathbf{first}_k^j(X) = \mathbf{first}_k^{j+1}(X) \text{ para todo } j \geq i$$

Teorema 8

Sea i^* el menor número tal que $\mathbf{first}_k^{i^*}(X) = \mathbf{first}_k^{i^*+1}(X)$ para todo $X \in V$. Entonces, para todo $X \in V$:

$$\mathbf{first}_k^{i^*}(X) = \mathbf{first}_k(X)$$

La demostración del teorema anterior queda como ejercicio propuesto para el lector. Una idea para la dirección \subseteq , es demostrar por inducción que $\mathbf{first}_k^i(X) \subseteq \mathbf{first}_k(X)$. Para la dirección \supseteq , una idea es demostrar por inducción que si $X \xRightarrow{*} w$, entonces $w|_k \in \mathbf{first}_k^i(X)$ para algún i .

Algoritmo. A continuación se presenta un algoritmo para calcular \mathbf{first}_k :

- ♦ **Input:** Gramática $\mathcal{G} = (V, \Sigma, P, S)$ y $k \geq 1$.
- ♦ **Output:** Todos los conjuntos $\mathbf{first}_k(X)$ para todo $X \in (V \cup \Sigma)$.

Function $\text{CalcularFirst}(\mathcal{G}, k)$:

```

foreach  $a \in \Sigma$  do
  |  $\text{first}_k^0(a) := \{a\}$ 
foreach  $X \in V$  do
  |  $\text{first}_k^0(X) := \cup_{X \rightarrow w \in P} w|_k$ 
 $i := 0$ 
repeat
  |  $i := i + 1$ 
  | foreach  $a \in \Sigma$  do
    |  $\text{first}_k^i(a) := \{a\}$ 
  | foreach  $X \in V$  do
    |  $\text{first}_k^i(X) := \bigcup_{X \rightarrow X_1 \dots X_n \in P} \text{first}_k^{i-1}(X_1) \odot_k \dots \odot_k \text{first}_k^{i-1}(X_n)$ 
until  $\text{first}_k^i(X) = \text{first}_k^{i-1}(X)$  para todo  $X \in (V \cup \Sigma)$ 
return  $\{\text{first}_k(X)\}_{X \in (V \cup \Sigma)}$ 

```

5.3.4. Calcular Follow

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$. Si consideramos $X \neq S$:

$$\begin{aligned}
 \text{follow}_k(X) &= \bigcup_{S \xrightarrow{*} \alpha X \beta} \text{first}_k(\beta\#) \\
 &= \bigcup_{S \xrightarrow{*} \alpha Y \beta \Rightarrow \alpha \alpha' X \beta' \beta} \text{first}_k(\beta' \beta\#) \\
 &= \bigcup_{Y \rightarrow \alpha' X \beta'} \bigcup_{S \xrightarrow{*} \alpha Y \beta} \text{first}_k(\beta' \beta\#) \\
 &= \bigcup_{Y \rightarrow \alpha' X \beta'} \bigcup_{S \xrightarrow{*} \alpha Y \beta} \text{first}_k(\beta') \odot_k \text{first}_k(\beta\#) \\
 &= \bigcup_{Y \rightarrow \alpha' X \beta'} \text{first}_k(\beta') \odot_k \bigcup_{S \xrightarrow{*} \alpha Y \beta} \text{first}_k(\beta\#) \\
 &= \bigcup_{Y \rightarrow \alpha' X \beta'} \text{first}_k(\beta') \odot_k \text{follow}_k(Y)
 \end{aligned}$$

Si consideramos $X = S$:

$$\begin{aligned}
 \text{follow}_k(S) &= \{\#\} \cup \bigcup_{S \xrightarrow{*} \alpha S \beta} \text{first}_k(\beta\#) \\
 &= \{\#\} \cup \bigcup_{S \xrightarrow{*} \alpha Y \beta \Rightarrow \alpha \alpha' S \beta' \beta} \text{first}_k(\beta' \beta\#) \\
 &= \{\#\} \cup \bigcup_{Y \rightarrow \alpha' S \beta'} \text{first}_k(\beta') \odot_k \text{follow}_k(Y)
 \end{aligned}$$

Dado lo anterior, podemos definir el siguiente teorema.

Teorema 9

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$. Entonces:

$$\text{Para } X \neq S: \quad \text{follow}_k(X) = \bigcup_{Y \rightarrow \alpha X \beta} \text{first}_k(\beta) \odot_k \text{follow}_k(Y)$$

$$\text{Para } X = S: \quad \text{follow}_k(S) = \{\#\} \cup \bigcup_{Y \rightarrow \alpha S \beta} \text{first}_k(\beta) \odot_k \text{follow}_k(Y)$$

Definimos el siguiente **programa recursivo** para todo $X \in V$:

$$\text{Para } X \neq S: \quad \text{follow}_k^0(X) := \emptyset$$

$$\text{Para } X = S: \quad \text{follow}_k^0(S) := \{\#\}$$

$$\text{Para } X \neq S: \quad \text{follow}_k^i(X) := \bigcup_{Y \rightarrow \alpha X \beta} \text{first}_k(\beta) \odot_k \text{follow}_k^{i-1}(Y)$$

$$\text{Para } X = S: \quad \text{follow}_k^i(S) := \{\#\} \cup \bigcup_{Y \rightarrow \alpha S \beta} \text{first}_k(\beta) \odot_k \text{follow}_k^{i-1}(Y)$$

Similar al caso de first_k , es fácil ver que:

♦ $\text{follow}_k^{i-1}(X) \subseteq \text{follow}_k^i(X)$ para todo $i > 1$.

♦ Como $\text{follow}_k(X) \subseteq \Sigma^{\leq k}$, entonces para algún $i \leq k \cdot |\Sigma|^k \cdot |V|$:

$$\text{follow}_k^j(X) = \text{follow}_k^{j+1}(X) \quad \text{para todo } j \geq i$$

Teorema 10

Sea i^* el menor número tal que $\text{follow}_k^{i^*}(X) = \text{follow}_k^{i^*+1}(X)$ para todo $X \in V$. Entonces, para todo $X \in V$:

$$\text{follow}_k^{i^*}(X) = \text{follow}_k(X)$$

La demostración de este teorema se deja como ejercicio propuesto al lector.

Con todo lo anterior, podemos calcular $\text{follow}_k(X)$ con un algoritmo similar que $\text{first}_k(X)$. Respecto a la eficiencia de este tipo de algoritmos:

♦ Toman $\mathcal{O}(k \cdot |\Sigma|^k \cdot |V|)$ en el peor caso.

♦ Si $k = 1$, el número de repeticiones será $\mathcal{O}(|\Sigma| \cdot |V|)$ y el tiempo del algoritmo será polinomial en $|\mathcal{G}|$ en el peor caso. Incluso, se puede hacer en tiempo $\mathcal{O}(|V| \cdot |P|)$ en total.

5.4. Gramáticas LL

Volvamos a la idea de buscar un algoritmo que haga parsing en **tiempo lineal**. Para esto, contruíamos un autómata apilador alternativo \mathcal{D} al cual le expandíamos sus producciones. ¿El problema? No sabemos como elegir que producciones expandir. Debido a lo anterior, introducimos los conceptos de **first** y **follow**. Así que, si tenemos una producción de la forma $X \rightarrow \alpha \mid \beta$, ¿cómo elegir entre α o β ?

Estrategia (intuición). La idea es la siguiente:

1. Mirar k símbolos del resto del input v (k -lookahead).
2. Usar $v|_k$ y decidir cuál regla $X \rightarrow \gamma$ elegimos para expandir.

La caracterización de las gramáticas que cumplen las propiedades anteriores se denominan **Gramáticas LL(k)**, donde

- ♦ **Primera L**: leer el input de izquierda a derecha (**Left-right**).
- ♦ **Segunda L**: producir una derivación por la izquierda (**Leftmost**).
- ♦ **Parámetro k** : el número de letras en adelante que utiliza (**lookahead**).

5.4.1. Definición Gramáticas LL

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto y $k \geq 1$.

Definición. Decimos que \mathcal{G} es una gramática LL(k) si para todas las derivaciones:

- ♦ $S \xRightarrow[\text{lm}]{*} uY\beta \Rightarrow_{\text{lm}} u\gamma_1\beta \xRightarrow[\text{lm}]{*} uv_1$
- ♦ $S \xRightarrow[\text{lm}]{*} uY\beta \Rightarrow_{\text{lm}} u\gamma_2\beta \xRightarrow[\text{lm}]{*} uv_2$ y
- ♦ $v_1|_k = v_2|_k$

entonces se cumple que $\gamma_1 = \gamma_2$.

Notar que la elección de $Y \rightarrow \gamma$ depende de $Y, v|_k$ y u .

Ejemplo 5.11: Gramáticas LL(1)

$$\mathcal{G}_1: S \rightarrow (S) \mid n$$

$$\begin{array}{ccccccc} S & \xRightarrow[\text{lm}]{*} & \overbrace{(\dots)}^u \overbrace{(S)}^Y \overbrace{\dots)}^\beta & \Rightarrow_{\text{lm}} & (\dots(\gamma_1)\dots) & \xRightarrow[\text{lm}]{*} & \overbrace{(\dots)}^u \overbrace{(v'_1)\dots)}^{v_1} \\ & & & \Rightarrow_{\text{lm}} & (\dots(\gamma_2)\dots) & \xRightarrow[\text{lm}]{*} & \underbrace{(\dots)}_u \underbrace{(v'_2)\dots}_{v_2} \end{array}$$

- ♦ Si $v_1|_1 = v_2|_1 = n$, entonces $\gamma_1 = \gamma_2 = n$.
- ♦ Si $v_1|_1 = v_2|_1 = ($, entonces $\gamma_1 = \gamma_2 = (S)$.

En ambos casos, tenemos que $\gamma_1 = \gamma_2$ y \mathcal{G}_1 es una gramática LL(1).

$$\begin{array}{l} \mathcal{G}_2: S \rightarrow (X) \mid n \\ X \rightarrow SX \mid \epsilon \end{array}$$

$$\begin{array}{ccccccc} S & \xRightarrow[\text{lm}]{*} & uX\beta & \Rightarrow_{\text{lm}} & u\gamma_1\beta & \xRightarrow[\text{lm}]{*} & uv_1 \\ & & & \Rightarrow_{\text{lm}} & u\gamma_2\beta & \xRightarrow[\text{lm}]{*} & uv_2 \end{array}$$

- ♦ Si $v_1|_1 = v_2|_1 = ($ o ' n ', entonces $\gamma_1 = \gamma_2 = SX$.
- ♦ Si $v_1|_1 = v_2|_1 =)$, entonces $\gamma_1 = \gamma_2 = \epsilon$.

Por lo tanto, tenemos que $\gamma_1 = \gamma_2$ y \mathcal{G}_2 es **también** una gramática LL(1).

Ejemplo 5.12: Gramática NO LL(1) pero si LL(2)

$$\begin{aligned} \mathcal{G}_3: \quad S &\rightarrow (X) \mid n + S \mid n \\ X &\rightarrow SX \mid \epsilon \end{aligned}$$

$$\begin{array}{ccccccc} S & \xRightarrow[\text{lm}]{*} & \overbrace{(S X)}^{u \ Y \ \beta} & \Rightarrow_{\text{lm}} & \overbrace{(n + S X)}^{\gamma_1} & \xRightarrow[\text{lm}]{*} & \overbrace{(n + n)}^{u \ v_1} \\ & & \Rightarrow_{\text{lm}} & & \overbrace{(n X)}^{\gamma_2} & \xRightarrow[\text{lm}]{*} & \overbrace{(n)}^{u \ v_2} \\ S & \xRightarrow[\text{lm}]{*} & u S \beta & \Rightarrow_{\text{lm}} & u \gamma_1 \beta & \xRightarrow[\text{lm}]{*} & u v_1 \\ & & \Rightarrow_{\text{lm}} & & u \gamma_2 \beta & \xRightarrow[\text{lm}]{*} & u v_2 \end{array}$$

Como $v_1|_1 = v_2|_1 = n$ pero $\gamma_1 \neq \gamma_2$, entonces \mathcal{G}_3 NO es una gramática LL(1).

- ♦ Si $v_1|_2 = v_2|_2 = n +$, entonces $\gamma_1 = \gamma_2 = n + S$.
- ♦ Si $v_1|_1 = v_2|_1 = na$, con $a \neq +$, entonces $\gamma_1 = \gamma_2 = n$.

Por lo tanto, tenemos que $\gamma_1 = \gamma_2$ y entonces \mathcal{G}_3 es LL(2).

Ejemplo 5.13: Gramática NO LL(k)

$$\begin{aligned} \mathcal{G}_4: \quad S &\rightarrow (X) \mid (X)^{\wedge e} \mid n + S \mid n \\ X &\rightarrow SX \mid \epsilon \end{aligned}$$

$$\begin{array}{ccccccc} S & \xRightarrow[\text{lm}]{*} & \overbrace{(S X)}^{u \ Y \ \beta} & \Rightarrow_{\text{lm}} & \overbrace{((S)^{\wedge e} X)}^{\gamma_1} & \xRightarrow[\text{lm}]{*} & \overbrace{((\dots^k(n)\dots^k)^{\wedge e})}^{u \ v_1} \\ & & \Rightarrow_{\text{lm}} & & \overbrace{((S) X)}^{\gamma_2} & \xRightarrow[\text{lm}]{*} & \overbrace{((\dots^k(n)\dots^k))}^{u \ v_2} \end{array}$$

Como $v_1|_k = v_2|_k = (\dots^k($ pero $\gamma_1 \neq \gamma_2$, entonces \mathcal{G}_4 NO es una gramática LL(k) para todo k .

Ejemplo 5.14: Gramática NO LL(k) transformada en LL(2)

La gramática \mathcal{G}_4 del ejemplo anterior se puede transformar para que sea LL(2) de la siguiente manera:

$$\begin{aligned} \mathcal{G}'_4: \quad S &\rightarrow (XY \mid n + S \mid n \\ X &\rightarrow SX \mid \epsilon \\ Y &\rightarrow) \mid)^{\wedge e} \end{aligned}$$

Queda como ejercicio para el lector demostrar que \mathcal{G}'_4 es LL(2).

Ejemplo 5.15: Lenguaje NO LL(k)

$$\begin{array}{ll}
\mathcal{G}_5: & S \rightarrow X \mid Y \\
& X \rightarrow aXb \mid 0 \\
& Y \rightarrow aYbb \mid 1
\end{array}
\quad
\mathcal{L}(\mathcal{G}_5) = \{a^n 0 b^n \mid n \geq 0\} \cup \{a^n 1 b^{2n} \mid n \geq 0\}$$

$$\begin{array}{ccccc}
S & \xRightarrow[\text{lm}]{*} & S & \Rightarrow & X & \xRightarrow[\text{lm}]{*} & a^k 0 b^k \\
& & & \Rightarrow & Y & \xRightarrow[\text{lm}]{*} & a^k 1 b^{2k}
\end{array}$$

Para todo $k \geq 1$, se tiene que \mathcal{G}_5 NO es una gramática LL(k).

Es posible demostrar que, para toda gramática \mathcal{G} con $\mathcal{L}(\mathcal{G}_5) = \mathcal{L}(\mathcal{G})$, \mathcal{G} NO es una gramática LL(k) para todo $k \geq 1$.

5.4.2. Caracterización LL

Para esta parte es importante manejar las definiciones de prefijos vistas en la sección 5.3.1.

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática libre de contexto **reducida** y $k \geq 1$. En base a esto definimos el siguiente teorema:

Teorema 11

\mathcal{G} es una gramática LL(k) si, y sólo si, para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ y para todo $S \xRightarrow[\text{lm}]{*} uY\beta$, se tiene que:

$$\text{first}_k(\gamma_1\beta) \cap \text{first}_k(\gamma_2\beta) = \emptyset$$

Demostración. (\Rightarrow) Por contrapositivo, supongamos que $v \in \text{first}_k(\gamma_1\beta) \cap \text{first}_k(\gamma_2\beta)$. Como \mathcal{G} es reducida (sin variables inútiles), entonces

$$\begin{array}{ccccccc}
S & \xRightarrow[\text{lm}]{*} & uY\beta & \Rightarrow & u\gamma_1\beta & \xRightarrow[\text{lm}]{*} & uvv_1 \\
& & & \Rightarrow & u\gamma_2\beta & \xRightarrow[\text{lm}]{*} & uvv_2
\end{array}$$

para algún $v_1, v_2 \in \Sigma^*$. Como $\gamma_1 \neq \gamma_2$, entonces \mathcal{G} NO es LL(k).

(\Leftarrow) Por contrapositivo (de nuevo), supongamos que \mathcal{G} no es LL(k). Como \mathcal{G} no es LL(k), entonces tenemos derivaciones de la forma:

$$\begin{array}{ccccccc}
S & \xRightarrow[\text{lm}]{*} & uY\beta & \Rightarrow & u\gamma_1\beta & \xRightarrow[\text{lm}]{*} & uv_1 \\
& & & \Rightarrow & u\gamma_2\beta & \xRightarrow[\text{lm}]{*} & uv_2
\end{array}$$

Vemos que $v_1|_k = v_2|_k = v$, pero $\gamma_1 \neq \gamma_2$. Por lo tanto, $v \in \text{first}_k(\gamma_1\beta) \cap \text{first}_k(\gamma_2\beta)$. ■

¿Cómo usamos la caracterización del teorema para demostrar que una gramática es LL(k)? Buscaremos condiciones más simples para verificar si una gramática es LL(k).

Definición. \mathcal{G} es una gramática LL(k) **fuerte** si para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ se tiene que:

$$\text{first}_k(\gamma_1) \odot_k \text{follow}_k(Y) \cap \text{first}_k(\gamma_2) \odot_k \text{follow}_k(Y) = \emptyset$$

Ejemplo 5.16: Si \mathcal{G} es $\text{LL}(k)$ fuerte, entonces \mathcal{G} es $\text{LL}(k)$

Una gramática \mathcal{G} que sea $\text{LL}(k)$ fuerte siempre es $\text{LL}(k)$, ya que si definimos dos conjuntos dados por el teorema de $\text{LL}(k)$ (F_1) y la definición de $\text{LL}(k)$ fuerte (F_2), dados por:

$$\begin{aligned} F_1 &= \text{first}_k(\gamma_1\beta) \cap \text{first}_k(\gamma_2\beta) = \text{first}_k(\gamma_1) \odot_k \text{first}_k(\beta) \cap \text{first}_k(\gamma_2) \odot_k \text{first}_k(\beta) \\ F_2 &= \text{first}_k(\gamma_1) \odot_k \text{first}_k(Y) \cap \text{first}_k(\gamma_2) \odot_k \text{first}_k(Y) = \emptyset \end{aligned}$$

Entonces, tenemos que $F_1 \subseteq F_2$.

Ejemplo 5.17: Si \mathcal{G} es $\text{LL}(k)$, ¿es $\text{LL}(k)$ fuerte?

La respuesta directa es que no. Con un contraejemplo, tomemos la gramática \mathcal{G} definida por

$$\begin{aligned} \mathcal{G}: \quad S &\rightarrow aXaa \mid bXba \\ X &\rightarrow b \mid \epsilon \end{aligned}$$

Recordatorio: \mathcal{G} es $\text{LL}(k)$ si para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ y para todo $S \xRightarrow[\text{lm}]{*} uY\beta$, se tiene que

$$\text{first}_k(\gamma_1\beta) \cap \text{first}_k(\gamma_2\beta) = \emptyset$$

♦ Si $S \xRightarrow[\text{lm}]{*} aXaa$, entonces $\text{first}_2(baa) \cap \text{first}_2(aa) = \emptyset$.

♦ Si $S \xRightarrow[\text{lm}]{*} bXba$, entonces $\text{first}_2(baa) \cap \text{first}_2(ba) = \emptyset$

Por lo tanto, \mathcal{G} es $\text{LL}(2)$.

Recordatorio: \mathcal{G} es una gramática $\text{LL}(k)$ fuerte si para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ se tiene que:

$$\text{first}_k(\gamma_1) \odot_k \text{follow}_k(Y) \cap \text{first}_k(\gamma_2) \odot_k \text{follow}_k(Y) = \emptyset$$

Si vemos $X \rightarrow b$ y $X \rightarrow \epsilon$:

$$\begin{aligned} &\text{first}_2(b) \odot_2 \text{follow}_2(X) \cap \text{first}_2(\epsilon) \odot_2 \text{follow}_2(X) \\ &= \{b\} \odot_2 \{aa, ba\} \cap \{\epsilon\} \odot_2 \{aa, ba\} \\ &= \{ba, bb\} \cap \{aa, ba\} \\ &= \{ba\} \quad \text{y por ende } \mathcal{G} \text{ no es } \text{LL}(2) \text{ fuerte.} \end{aligned}$$

¿Qué pasa con el caso $\text{LL}(1)$? Supongamos que \mathcal{G} es $\text{LL}(1)$ y $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$ son reglas distintas.

1. Si $\epsilon \notin \text{first}_1(\gamma_1)$ y $\epsilon \notin \text{first}_1(\gamma_2)$, entonces, por la caracterización de $\text{LL}(1)$:

$$\begin{aligned} \emptyset &= \text{first}_1(\gamma_1\beta) \cap \text{first}_1(\gamma_2\beta) \\ &= \text{first}_1(\gamma_1) \cap \text{first}_1(\gamma_2) \\ &= \text{first}_1(\gamma_1) \odot_1 \text{follow}_1(Y) \cap \text{first}_1(\gamma_2) \odot_1 \text{follow}_1(Y) \end{aligned}$$

2. Si $\epsilon \in \text{first}_1(\gamma_1)$ y $\epsilon \notin \text{first}_1(\gamma_2)$, entonces, por la caracterización de $\text{LL}(1)$:

$$\begin{aligned} \emptyset &= \text{first}_1(\gamma_1\beta) \cap \text{first}_1(\gamma_2\beta) \\ &= \text{first}_1(\gamma_1\beta) \cap \text{first}_1(\gamma_2) \\ &= \text{first}_1(\gamma_1\beta) \cap \text{first}_1(\gamma_2\beta') \end{aligned}$$

para todo $\beta' \in (V \cup \Sigma)^*$. Por lo tanto:

$$\begin{aligned} & \text{first}_1(\gamma_1) \odot_1 \text{follow}_1(Y) \cap \text{first}_1(\gamma_2) \odot_1 \text{follow}_1(Y) \\ &= \bigcup_{S \xrightarrow[\text{lm}]{\neq} uY\beta} \text{first}_1(\gamma_1\beta) \cap \bigcup_{S \xrightarrow[\text{lm}]{\neq} uY\beta'} \text{first}_1(\gamma_2\beta') = \emptyset \end{aligned}$$

Por lo tanto, establecemos el siguiente teorema.

Teorema 12

Una gramática \mathcal{G} es $LL(1)$ si, y sólo si, \mathcal{G} es $LL(1)$ **fuerte**, esto es, para todas dos reglas distintas $Y \rightarrow \gamma_1, Y \rightarrow \gamma_2 \in P$:

$$\text{first}_1(\gamma_1) \odot_1 \text{follow}_1(Y) \cap \text{first}_1(\gamma_2) \odot_1 \text{follow}_1(Y) = \emptyset$$

La condición del teorema anterior se puede verificar en **tiempo polinomial** en \mathcal{G} .

5.5. Parsing con gramáticas $LL(k)$

5.5.1. Algunas consideraciones

Considere la siguiente gramática \mathcal{G} :

$$\begin{aligned} S &\rightarrow Xa \mid Xb \\ X &\rightarrow c \end{aligned}$$

¿Es esta gramática del tipo $LL(1)$? Podemos ver que $\text{first}_1(\gamma_1\beta) = \{c\}$ y $\text{first}_1(\gamma_2\beta) = \{c\}$, con $\gamma_1 = Xa$, $\gamma_2 = Xb$ y $\beta = \epsilon$. Por lo tanto su intersección no es vacía y entonces \mathcal{G} no es $LL(1)$. ¿Podemos establecer una solución para este problema?

Factorización. En general, si tenemos una regla:

$$X \rightarrow \gamma\alpha_1 \mid \gamma\alpha_2$$

siempre podemos “**factorizar**” la regla manteniendo la semántica, como:

$$\begin{aligned} X &\rightarrow \gamma X' \\ X' &\rightarrow \alpha_1 \mid \alpha_2 \end{aligned}$$

Considere ahora la siguiente gramática \mathcal{G} :

$$E \rightarrow E * E \mid n$$

¿Es esta gramática del tipo $LL(1)$? ¿ $LL(k)$? Pues no es ninguna. El problema con esta gramática es su recursividad, en específico, por la izquierda.

Definición. Una gramática \mathcal{G} se dice **recursiva por la izquierda** si existe $X \in V$ tal que:

$$X \xRightarrow{+} X\gamma \quad \text{para algún } \gamma \in (V \cup \Sigma)^*$$

Teorema 13

Si $\mathcal{G} = (V, \Sigma, P, S)$ es una gramática reducida y recursiva por la izquierda, entonces \mathcal{G} NO es $LL(k)$ para todo $k \geq 1$.

Demostración. Por simplicidad, suponga que $X \rightarrow X\beta \in P$ y $X \rightarrow w \in P$.

Como \mathcal{G} es reducida, entonces existe una derivación $S \xRightarrow[\text{lm}]{*} uX\gamma$:

$$S \xRightarrow[\text{lm}]{*} uX\gamma \xRightarrow[\text{lm}]{n\text{-veces}} uX\beta^n\gamma$$

Por **contradicción**, suponga que \mathcal{G} es $\text{LL}(k)$. Por lo tanto:

$$\text{first}_k(X\beta^{n+1}\gamma) \cap \text{first}_k(w\beta^n\gamma) = \emptyset$$

Suponga que $\beta \xRightarrow{*} v \in \Sigma^*$ y $\gamma \xRightarrow{*} v' \in \Sigma^*$. Con $n = k$, tendremos que

$$(wv^kv')|_k \in \text{first}_k(X\beta^{k+1}\gamma) \cap \text{first}_k(w\beta^k\gamma) \rightarrow \leftarrow \text{ (¡contradicción! el conjunto no es vacío)}$$

■

Hablemos de recursión **inmediata** por la izquierda. Suponga que existe $X \in V$ tal que:

$$X \rightarrow X\alpha_1 \mid \cdots \mid X\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$

¿Cómo podemos **eliminar** la recursión inmediata por la izquierda? Consideramos la misma gramática pero **cambiando** las reglas de X por:

$$\begin{aligned} X &\rightarrow \beta_1 X' \mid \cdots \mid \beta_n X' \\ X' &\rightarrow \alpha_1 X' \mid \cdots \mid \alpha_m X' \mid \epsilon \end{aligned}$$

Ejemplo 5.18: Eliminando recursión inmediata

$$\begin{array}{lcl} S & \rightarrow & Xa \mid b \\ X & \rightarrow & Xc \mid d \end{array} \quad \left| \quad \begin{array}{lcl} S & \rightarrow & Xa \mid b \\ X & \rightarrow & dX' \\ X' & \rightarrow & cX' \mid \epsilon \end{array}$$

Teorema 14

Sea \mathcal{G} una gramática tal que existe $X \in V$:

$$X \rightarrow X\alpha_1 \mid \cdots \mid X\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$

Sea \mathcal{G}' la misma gramática \mathcal{G} pero cambiando las reglas de X por:

$$\begin{aligned} X &\rightarrow \beta_1 X' \mid \cdots \mid \beta_n X' \\ X' &\rightarrow \alpha_1 X' \mid \cdots \mid \alpha_m X' \mid \epsilon \end{aligned}$$

Entonces $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$

Demostración. Una derivación por la izquierda de X en \mathcal{G} :

$$X \xRightarrow[\text{lm}]{} X\alpha_{i_1} \xRightarrow[\text{lm}]{} X\alpha_{i_2}\alpha_{i_1} \xRightarrow[\text{lm}]{} \cdots \xRightarrow[\text{lm}]{} X\alpha_{i_p}\alpha_{i_{p-1}}\cdots\alpha_{i_1} \xRightarrow[\text{lm}]{} \beta_j\alpha_{i_p}\alpha_{i_{p-1}}\cdots\alpha_{i_1}$$

Una derivación por la derecha de X en \mathcal{G}' **equivalente**:

$$X \xRightarrow[\text{rm}]{} \beta_j X' \xRightarrow[\text{rm}]{} \beta_j \alpha_{i_p} X' \xRightarrow[\text{rm}]{} \cdots \xRightarrow[\text{rm}]{} \beta_j \alpha_{i_p} \cdots \alpha_{i_2} \alpha_{i_1} X' \xRightarrow[\text{rm}]{} \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \cdots \alpha_{i_1}$$

■

Ahora, ¿qué pasa si la recursión por la izquierda es **no-inmediata**? Considere la siguiente gramática **recursiva por la izquierda**:

$$S \rightarrow Xa \mid b$$

$$X \rightarrow Yc$$

$$Y \rightarrow Xd \mid e$$

¿Cómo eliminamos la recursión por la izquierda no-inmediata?

Estrategia. Dado $V = \{X_1, \dots, X_n\}$, removemos la recursión inductivamente en n , tal que, en cada paso i de la inducción, se cumpla que para todo $i, j \leq n$:

$$\text{si } X_i \rightarrow X_j \alpha, \text{ entonces } i < j$$

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$ y $V = \{X_1, \dots, X_n\}$

output: Gramática \mathcal{G} sin recursión por la izquierda

Function EliminarRecursion(\mathcal{G}):

$P' := P$

for $i = 1$ **to** n **do**

for $j = 1$ **to** $i - 1$ **do**

foreach $X_i \rightarrow X_j \gamma \in P'$ **do**

foreach $X_j \rightarrow \alpha \in P'$ **do**

$P' := P' \cup \{X_i \rightarrow \alpha \gamma\}$

$P' := P' - \{X_i \rightarrow X_j \gamma\}$

Remover recursión inmediata para X_i en P' (si existe)

$V' := \{X_1, \dots, X_n\} \cup \{X'_1, \dots, X'_n\}$

return (V', Σ, P', S)

Queda como ejercicio propuesto al lector demostrar la correctitud del algoritmo.

Ejemplo 5.19: Eliminando recursión

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid n$$

Eliminando la **recursión inmediata** de E :

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid n$$

Eliminando la **recursión inmediata** de T :

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid n$$

Conclusión. Es posible eliminar la recursividad por la izquierda, pero esto **NO asegura** que el resultado sea una gramática $LL(k)$ para algún k .

5.5.2. Parsing de $LL(k)$

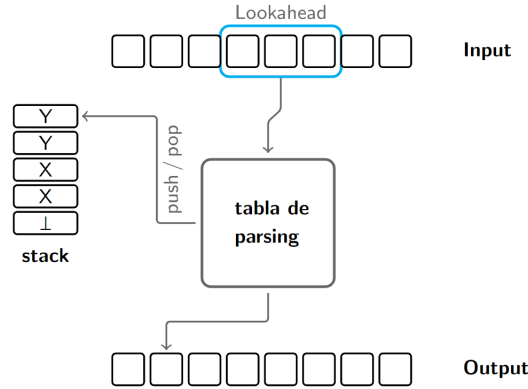


Figura 6: Máquina de parsing

Definición. Sea Σ un alfabeto finito. Se definen los siguientes conjuntos de palabras:

- ♦ $\dot{\Sigma} = \Sigma^* \times \Sigma^*$
- ♦ $\dot{\Sigma}^{\leq k} = \{(u, v) \in \dot{\Sigma} \mid |uv| \leq k\}$
- ♦ $\dot{\Sigma}_{\#}^{\leq k} = \{(u, v) \in \dot{\Sigma} \mid |uv| \leq k\} \cup \{(u, v\#) \mid (u, v) \in \dot{\Sigma} \mid |uv| < k\}$

Notación. En vez de usar $(u, v) \in \dot{\Sigma}_{\#}^{\leq k}$, escribiremos $u.v \in \dot{\Sigma}_{\#}^{\leq k}$. El par $\epsilon.\epsilon$ lo denotaremos solamente por ϵ .

Definición. Un transductor apilador con k -lookahead (k -PDT) es una tupla:

$$\mathcal{T} = (Q, \Sigma, \Omega, \Delta, q_0, F)$$

- ♦ Q es un conjunto finito de estados.
- ♦ Σ es el alfabeto de input.
- ♦ Ω es el alfabeto de output.
- ♦ $\Delta \subseteq Q^+ \times \dot{\Sigma}_{\#}^{\leq k} \times (\Omega \cup \{\epsilon\}) \times Q^*$ es la relación de transición.
- ♦ $q_0 \in Q$ es un conjunto de estados iniciales.
- ♦ $F \subseteq Q$ es el conjunto de estados finales.

Definición. Una **configuración** de \mathcal{T} es una tupla:

$$(q_1 \dots q_k, w, o) \in (Q^+, \Sigma^* \cdot \{\#\}, \Omega^*)$$

♦ $q_1 \dots q_k$ es el contenido del stack con q_1 el tope del stack.

♦ w es el contenido del input.

♦ o es el contenido del output.

Decimos que una configuración:

♦ $(q_0, w\#, \epsilon)$ es **inicial** y

♦ $(q_f, \#, o)$ es **final** si $q_f \in F$.

Definición. Se define la relación $\vdash_{\mathcal{T}}$ de **siguiente-paso** entre configuraciones de \mathcal{T} :

$$(\gamma_1, w_1, o_1) \vdash_{\mathcal{T}} (\gamma_2, w_2, o_2)$$

si, y sólo si, existe $(\alpha, u.v, a, \beta) \in \Delta$, $\gamma \in \Gamma^*$ y $w \in \Sigma^* \cdot \{\#\}$ tal que:

♦ **Stack:** $\gamma_1 = \alpha \cdot \gamma$ y $\gamma_2 = \beta \cdot \gamma$

♦ **Look-ahead:** $w_1 = u \cdot v \cdot w$ y $w_2 = v \cdot w$

♦ **Output:** $o_2 = o_1 \cdot a$

Se define $\vdash_{\mathcal{T}}^*$ como la clausura **refleja y transitiva** de $\vdash_{\mathcal{T}}$.

Definición. \mathcal{T} **entrega** o con **input** w si existe una configuración inicial $(q_0, w \cdot \#, \epsilon)$ y una configuración final $(q_f, \#, o)$ tal que:

$$(q_0, w \cdot \#, \epsilon) \vdash_{\mathcal{T}}^* (q_f, \#, o)$$

Se define la función $\llbracket \mathcal{T} \rrbracket : \Sigma^* \rightarrow 2^{\Omega^*}$:

$$\llbracket \mathcal{T} \rrbracket(w) = \{o \in \Omega^* \mid \mathcal{T} \text{ entrega } o \text{ con input } w\}$$

Definición. \mathcal{T} es **determinista** si para todo $(\alpha_1, u_1.v_1, a_1, \beta_1), (\alpha_2, u_2.v_2, a_2, \beta_2) \in \Delta$ con $(\alpha_1, u_1.v_1, a_1, \beta_1) \neq (\alpha_2, u_2.v_2, a_2, \beta_2)$ se cumple que

$$\alpha_1 \text{ NO es prefijo de } \alpha_2 \quad \text{o} \quad u_1.v_1 \text{ NO es prefijo de } u_2.v_2.$$

“Para cualquier configuración (γ, w, o) existe **a lo más** una configuración (γ', w', o) tal que $(\gamma, w, o) \vdash_{\mathcal{T}}^* (\gamma', w', o)$ ”

La **ventaja** de un k -PDT determinista es que nos aseguramos de que siempre obtenemos un solo output para cada input (el no-determinismo nos podría generar muchos outputs distintos).

Construcción del parser. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática $\text{LL}(k)$ fuerte. Se define el k -PDT para \mathcal{G} :

$$\mathcal{T}[\mathcal{G}] = \left(V \cup \Sigma \cup \{q_0, q_f\}, \Sigma, \underbrace{P}_{\Omega}, \Delta, q_0, \{q_f\} \right)$$

La relación de transición Δ se define como:

Inicio: $(q_0, \epsilon, \epsilon, S \cdot q_f)$

Reducir: $(a, a., \epsilon, \epsilon)$ para cada $a \in \Sigma$

Expandir: $(X, .u, p, \gamma)$

para cada $p := (X \rightarrow \gamma) \in P$ tal que $u \in \text{first}_k(\gamma) \odot_k \text{follow}_k(X)$

Propiedades. $\mathcal{T}[\mathcal{G}]$ tiene las siguientes propiedades:

1. $\mathcal{T}[\mathcal{G}]$ es un k -PDT **determinista** si, y sólo si, \mathcal{G} es $\text{LL}(k)$ fuerte.
2. Si $w \notin \mathcal{L}(\mathcal{G})$ entonces $\llbracket \mathcal{T} \rrbracket(w) = \emptyset$.
3. Si $w \in \mathcal{L}(\mathcal{G})$ entonces $\llbracket \mathcal{T} \rrbracket(w) = \{r_1 \dots r_m\}$ es una derivación por la izquierda de \mathcal{G} sobre w .

Algoritmo. Para una gramática $\text{LL}(k)$ \mathcal{G} y una palabra $w \in \Sigma^*$:

1. Construya el k -PDT determinista $\mathcal{T}[\mathcal{G}]$ a partir de \mathcal{G} .
2. Ejecute $\mathcal{T}[\mathcal{G}]$ sobre w .

Como $\mathcal{T}[\mathcal{G}]$ es determinista, entonces el algoritmo toma **tiempo lineal** en w .

Tabla predictiva para $\text{LL}(k)$ fuerte. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática $\text{LL}(k)$ fuerte. Para cada $u \in \Sigma^k \cup \Sigma^{<k} \cdot \{\#\}$, se define $M[X, u] \in (V \cup \Sigma)^* \cup \{\text{ERROR}\}$:

$$M[X, u] = \begin{cases} \gamma & \text{si } X \rightarrow \gamma \in P \text{ y } u \in \text{first}_k(\gamma) \odot_k \text{follow}_k(X) \\ \text{ERROR} & \text{en otro caso.} \end{cases}$$

El computo de la tabla predictiva puede tomar **tiempo exponencial** en $|\mathcal{G}|$ y k .

Caso especial: tabla predictiva para $\text{LL}(1)$. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una gramática $\text{LL}(1)$ fuerte. Para cada $a \in \Sigma \cup \{\#\}$, se define $M[x, a] \in (V \cup \Sigma)^* \cup \{\text{ERROR}\}$:

$$M[X, a] = \begin{cases} \gamma & \text{si } X \rightarrow \gamma \in P \text{ y } a \in \text{first}_1(\gamma) \\ \gamma & \text{si } X \rightarrow \gamma \in P, \epsilon \in \text{first}_1(\gamma) \text{ y } a \in \text{follow}_1(X) \\ \text{ERROR} & \text{en otro caso.} \end{cases}$$

Este cálculo se puede hacer en tiempo $\mathcal{O}(|V| \cdot |P|)$.

Ejemplo 5.20: Tabla predictiva

E	$\rightarrow TE'$	$\text{first}_1(E) = \{ (, \text{id} \}$	$\text{first}_1(E') = \{ +, \epsilon \}$
E'	$\rightarrow +TE' \mid \epsilon$	$\text{first}_1(T) = \{ (, \text{id} \}$	$\text{first}_1(T') = \{ *, \epsilon \}$
T	$\rightarrow FT'$	$\text{first}_1(F) = \{ (, \text{id} \}$	
T'	$\rightarrow *FT' \mid \epsilon$	$\text{follow}_1(E) = \{ \}, \# \}$	$\text{follow}_1(E') = \{ \}, \# \}$
F	$\rightarrow (E) \mid \text{id}$	$\text{follow}_1(T) = \{ +, \}, \# \}$	$\text{follow}_1(T') = \{ +, \}, \# \}$
		$\text{follow}_1(F) = \{ +, *, \}, \# \}$	

	id	+	*	()	#
E	TE'	ERROR	ERROR	TE'	ERROR	ERROR
E'	ERROR	$+TE'$	ERROR	ERROR	ϵ	ϵ
T	FT'	ERROR	ERROR	FT'	ERROR	ERROR
T'	ERROR	ϵ	$*FT'$	ERROR	ϵ	ϵ
F	id	ERROR	ERROR	(E)	ERROR	ERROR

6. Extracción de información

6.1. Extracción

6.2. Enumeración de resultados: Autómatas con anotaciones