Ignacio Méndez Pérez
IIC2223 – 2022 - 2

Interrogación 2

¿Cómo evaluamos un autómata no-determinista?

NFA *on-the-fly*:

Para un autómata no-determinista $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$:

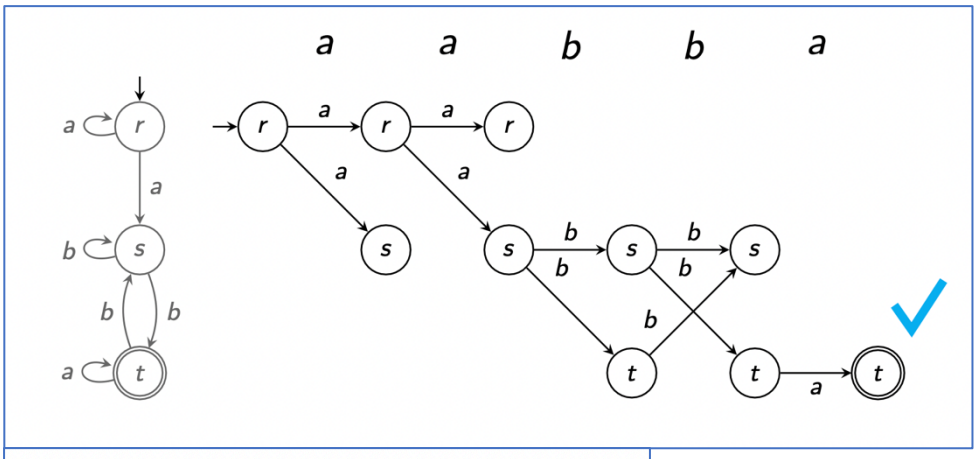
- $2^Q = \{S \mid S \subseteq Q\}$ es el conjunto potencia de Q.
- $q_0^{det} = I$.
- $\delta^{det}: 2^Q \times \Sigma \to 2^Q$ tal que:

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

- $F^{det} = \{S \in 2^Q \mid S \cap F \neq \emptyset\}$.

Estrategia:

- Mantenemos un conjunto S de estados actuales.
- Por cada nueva letra a, calculamos el conjunto $\delta^{det}(S, a)$.



Function `eval-NFAonthefly` $(\mathcal{A}, w)$

```
S := I
for i = 1 to n do
    S_old := S
    S := ∅
    foreach p ∈ S_old do
        S := S ∪ {q | (p, a_i, q) ∈ Δ}
return check (S ∩ F ≠ ∅)
```

Complejidad: $\mathcal{O}(|\mathcal{A}| \cdot |w|)$

ϵ-NFA *on-the-fly*:

```
Function eval-eNFAonthefly (A, w)
    S := ε-closure(Δ, I)
    for i = 1 to n do
        S_old := S
        S := ∅
        foreach p ∈ S_old do
            S := S ∪ {q | (p, a_i, q) ∈ Δ}
        S := ε-closure(Δ, S)
    return check(S ∩ F ≠ ∅)
```

Complejidad: $\mathcal{O}(|\mathcal{A}| \cdot |w|)$

Transductores

Un transductor (en inglés, transducer) es una tupla:

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

- Q es un conjunto finito de estados.
- Σ es el alfabeto del input.
- Ω es el alfabeto del **output**.
- $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times Q$ es la relación de transición.
- $I \subseteq Q$ es el conjunto de estados iniciales.
- $F \subseteq Q$ es el conjunto de estados finales.

"*Intuitivamente, una configuración (q, au, vb) representa que $\mathcal{T}$ se encuentra en el estado q, procesando la palabra au y leyendo a, y hasta ahora grabó la palabra vb y el último símbolo impreso es b.*"

Definición

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

$$(p_1, u_1, v_1) \vdash_{\mathcal{T}} (q, u_2, v_2)$$

Si, y sólo si, existe $(p, a, b, q) \in \Delta$ tal que $u_1 = a \cdot u_2$ y $v_2 = v_1 \cdot b$

Definiciones

- $\mathcal{T}$ **entrega** v con **input** u si existe una configuración **inicial** (q_0, u, ϵ) y una configuración **final** (q_f, ϵ, v) tal que:

$$(q_0, u, \epsilon) \vdash_{\mathcal{T}}^{*} (q_f, \epsilon, v)$$

- Se define la función $[\![\mathcal{T}]\!]: \Sigma^* \to 2^{\Omega^*}$:

$$[\![\mathcal{T}]\!](u) = \{v \in \Omega^* \mid \mathcal{T} \text{ entrega } v \text{ con input } u\}$$

$[\![\mathcal{T}]\!]$ → Semántica

- Se dice que $f: \Sigma^* \to 2^{\Omega^*}$ es una **función racional** si existe un transductor $\mathcal{T}$ tal que $f = [\![\mathcal{T}]\!]$.

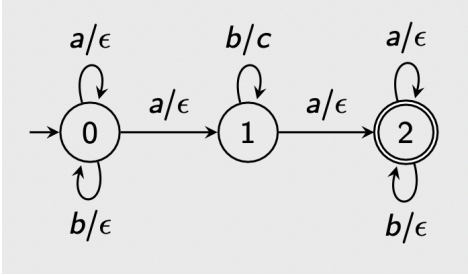
Un transductor define una función de palabras a conjunto de palabras.

Definiciones

Para una relación $R \subseteq \Sigma^* \times \Omega^*$ se define:

- $\pi_1(R) = \{ u \in \Sigma^* \mid \exists v \in \Omega^*. (u, v) \in R \} \Longrightarrow$ "Dominio" de R.
- $\pi_2(R) = \{ v \in \Omega^* \mid \exists u \in \Sigma^*. (u, v) \in R \} \Longrightarrow$ "Recorrido" de R.

¿Cuál es el lenguaje definido por $\pi_1([\![\mathcal{T}]\!])$ y $\pi_2([\![\mathcal{T}]\!])$?



Dominio: Todas las palabras con 2 o más a's

Recorrido: c^*

Teorema

Sea $\mathcal{T}_1$ y $\mathcal{T}_2$ dos transductores con Σ y Ω alfabetos de input y output.
Las siguientes son **relaciones racionales**.

1. $[\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!] = \{(u, v) \in \Sigma^* \times \Omega^* \mid (u, v) \in [\![\mathcal{T}_1]\!] \vee (u, v) \in [\![\mathcal{T}_2]\!]\}$.
2. $[\![\mathcal{T}_1]\!] \cdot [\![\mathcal{T}_2]\!] = \{(u_1 u_2, v_1 v_2) \in \Sigma^* \times \Omega^* \mid (u_1, v_1) \in [\![\mathcal{T}_1]\!] \wedge (u_2, v_2) \in [\![\mathcal{T}_2]\!]\}$
3. $[\![\mathcal{T}_1]\!]^* = \bigcup_{k=0}^{\infty} [\![\mathcal{T}_1]\!]^k$

Teorema

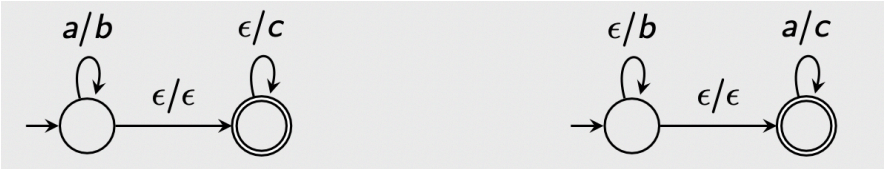
Existen transductores $\mathcal{T}_1$ y $\mathcal{T}_2$ sobre Σ y Ω tal que:

$$[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \{(u, v) \in \Sigma^* \times \Omega^* \mid (u, v) \in [\![\mathcal{T}_1]\!] \wedge (u, v) \in [\![\mathcal{T}_2]\!]\}$$

NO es una relación racional.

Demostración

Considere los siguientes transductores:



$[\![\mathcal{T}_1]\!] = \{(a^n, b^n c^m) \mid n \geq 0, m \geq 0\}.$ $[\![\mathcal{T}_2]\!] = \{(a^n, b^m c^n) \mid n \geq 0, m \geq 0\}$

Pero $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \{(a^n, b^n c^n) \mid n \geq 0\}$, y, por lo tanto, $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!]$ no es racional.
(El recorrido de la intersección es $\{b^n \cdot c^n\}$ el cual no es racional).

Definición

Decimos que un transductor $\mathcal{T}$ define una función (parcial) si:

$$\text{para todo } u \in \Sigma^* \text{ se tiene que } |[\![\mathcal{T}]\!](u)| \leq 1.$$

Definición

Decimos que $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ es **determinista** si cumple que:

1. $\mathcal{T}$ define una función $[\![\mathcal{T}]\!]: \Sigma^* \to \Omega^*$.
2. Para todo $(p, a_1, b_1, q_1) \in \Delta$ y $(p, a_2, b_2, q_2) \in \Delta$, si $a_1 = a_2$, entonces $b_1 = b_2$ y $q_1 = q_2$, es decir, son la misma transición.
3. Si $(p, \epsilon, b, q) \in \Delta$, entonces para todo $(p, a', b', q') \in \Delta$, se tien que $(a', b', q') = (\epsilon, b, q)$. Es decir, si tengo alguna ϵ transición, esta debe ser la única transición que sale de ese estado.

Sintaxis y semántica de un lenguaje de programación

Definición

1. La **sintaxis** de un lenguaje es un conjunto de reglas que describen los programas válidos que tienen significado.

2. La **semántica** de un lenguaje define el significado de un programa correcto según la sintaxis.

Verificación de sintaxis

En este proceso se busca:

- Verificar la sintaxis de un programa.
- Entregar la estructura de un programa (árbol de parsing).

Consta de 3 etapas:

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

Análisis léxico (LEXER)

- El análisis léxico consta en dividir el programa en una secuencia de **tokens**.
- Un **token** (o lexema) es un substring (válido) dentro de un programa.
- Un **token** está compuesto por:
    - Tipo
    - Valor (el valor mismo del substring).

Tipos usuales de **tokens** en lenguajes de programación:

- **number** (constante): 2, 235, 495, …
- **string** (constante): “hello”, “ilove TDA” ,…
- **keywords**: if, for, …
- **identificadores**: pos, init, rate, …
- **delimitadores**: “{“, “}”, “(“, “)”, “,”, …
- **operadores**: “=”, “+”, “<”, “<=”, …

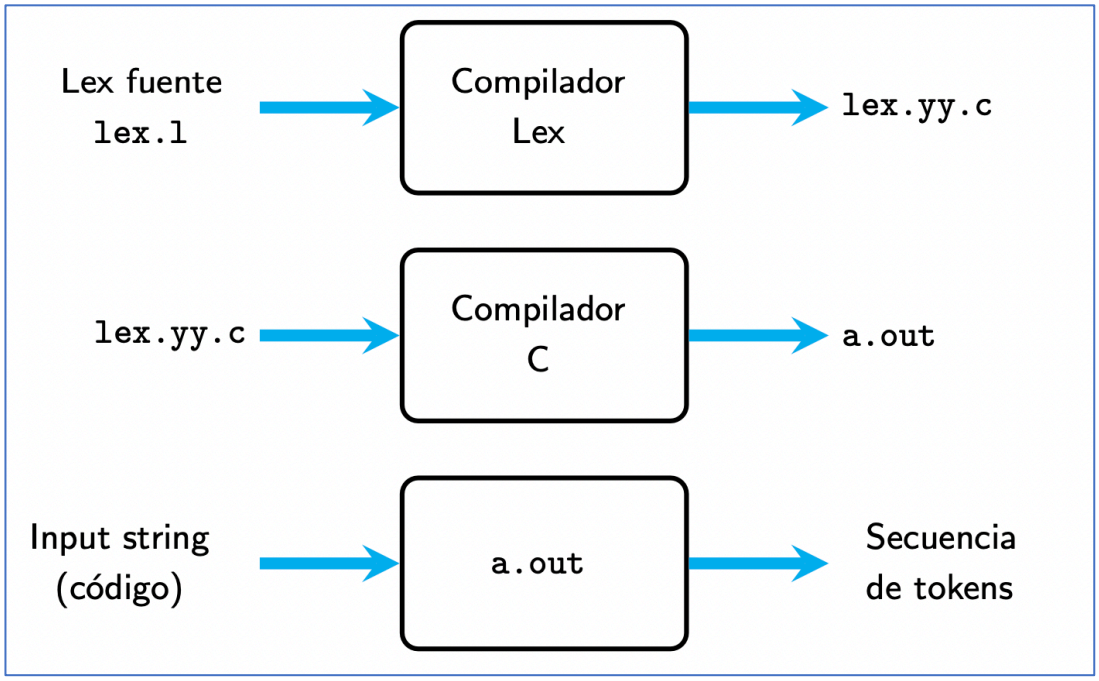
```
pos = init + rate * 60
```

Tipo	Valor
id	pos
EQ	=
id	init
PLUS	+
id	rate
MULT	*
number	60

Generador de análisis léxico (Lex)

- Un generador de análisis léxico es un software que, a través de un programa fuente, crea el código necesario para hacer el análisis léxico.

- Lex es el analizador léxico estándar en Unix en el lenguaje C.



El formato de un programa en Lex es de la forma:

```
declaraciones
%%
reglas de traducción
%%
funciones auxiliares
```

Ejemplo declaraciones:

```
%{
#include "misconstantes.h" \* def de IF, ELSE, ID, NUMBER *\
%}
delim   [\t\n]
ws      {delim}+
%%
...
```

Ejemplo funciones auxiliares:

```
...
%%
void printID(){printf("Id:  %s\n", yytext);}
void printNumebr(){printf("Number:    %s\n", yytext);}
```

Las **reglas de traducción** tienen la siguiente forma:

Patrón { Acción }

- **Patrón** está definido por una **expresión regular**.
- **Acción** es código C embebido.

Ejemplo: reglas de traducción:

```
[ \t\n]+                    {\* sin accion *\}
if                          { return(IF);}
else                        { return(ELSE);}
[A-Za-z][A-Za-z0-9]*        {printID(); return(ID);}
[0-9]+                      {printNumber(); return(NUMBER);}
```

Resolución de conflictos en Lex

Si **varios prefijos** del input satisfacen uno o más patrones:

1. Se prefiere el **prefijo más largo** por sobre el prefijo más corto.
2. Si el prefijo más corto satisface uno o más patrones, se prefiere **el patrón listado primero** en el programa lex.1.

¿Cómo evaluamos los patrones en lex.1?

Sea $P_1, \dots, P_k$ los patrones y $C_1, \dots, C_k$ las acciones en el programa "lex.1", respectivamente.

Primer paso

Para cada patrón P_i construimos un NFA $\mathcal{A}_i = \left(Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\}\right)$ con un solo estado inicial q_0^i y un solo estado final q_f^i.

Supondremos las siguientes **simplificaciones**:

1. Cada lexema esta separado por un símbolo de espacio "_".
2. Documento termina con un símbolo especial EOF.
3. No hay conflictos entre patrones.

Ejemplo conflictos:

```
if                          { return(IF);}
else                        { return(ELSE);}
[A-Za-z]([A-Za-z0-9])*      {printID(); return(ID);}
```

¿Cómo evaluamos los patrones en paralelo?

- $\mathcal{A}_i = \left(Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\}\right)$ el NFA para el **patrón** P_i y
- C_i la **acción** de P_i con $i \leq k$.

Evaluamos el transductor determinista:

$$\mathcal{T} = (Q, \Sigma, \{C_i\}_{i \leq k}, \Delta, \{q_0\}, F)$$

- $Q = 2^{\{\cup_{i=1}^{k} Q_i\}}$
- $q_0 = \{q_0^1, q_0^2, \ldots, q_0^k\}$
- $(S, a, \epsilon, S') \in \Delta$ si, y solo si, $S' = \{q \mid \exists i. \exists p \in S. (p, a, q) \in \Delta_i\}$
- $(S, \_, C_i, q_o), (S, EOF, C_i, q_0) \in \Delta$ si, y solo si, $q_f^i \in S$.
- $F = \{q_0\}$

Lexer on-the-fly (simplificado)

Sea $\mathcal{A}_i = \left(Q_i, \Sigma, \Delta_i, \{q_0^i\}, \{q_f^i\}\right)$ el NFA para el patrón P_i y $w = a_1 \ldots a_n$.

Function `lexer-onthefly` $(\mathcal{A}_1, \ldots, \mathcal{A}_k, w)$
- $S := \{q_0^1, q_0^2, \ldots, q_0^k\}$
- **for** $j = 1$ **to** n **do**
  - **if** $a_j \neq \sqcup$ **and** $a_j \neq EOF$ **then**
    - $S_{\text{old}} := S$
    - $S := \varnothing$
    - **foreach** $i = 1$ **to** k **do**
      - **foreach** $p \in S_{old} \cap Q_i$ **do**
        - $S := S \cup \{q \mid (p, a_i, q) \in \Delta_i\}$
  - ...

Function `lexer-onthefly` $(\mathcal{A}_1, \ldots, \mathcal{A}_k, w)$
- $S := \{q_0^1, q_0^2, \ldots, q_0^k\}$
- **for** $j = 1$ **to** n **do**
  - ...
  - **else if** $a_j = \sqcup$ **or** $a_j = EOF$ **then**
    - **if** $q_f^1 \in S$ **then**
      - **execute** C_1
    - ...
    - **else if** $q_f^k \in S$ **then**
      - **execute** C_k
    - **else**
      - **ERROR**
    - $S := \{q_0^1, q_0^2, \ldots, q_0^k\}$
- **return**

Tiempo análisis léxico

Si $|P_i|$ es el tamaño del patrón P_i:

$$\mathcal{O}((|P_1| + \cdots + |P_k|) \cdot |w|)$$

Algunas conclusiones/observaciones

1. El análisis léxico es equivalente a evaluar un **transductor** que simula los patrones en **paralelo**.
2. El análisis léxico también maneja **conflictos entre reglas** y otros detalles.

Algoritmo de Knuth-Morris-Prat

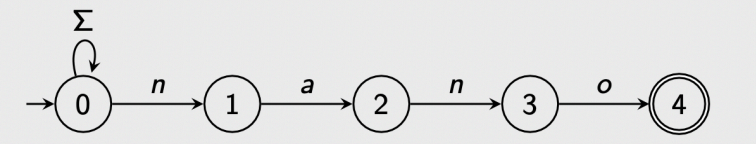
Problema

Dado un patrón $w = w_1 \dots w_m$ y un documento $d = d_1 \dots d_n$, encontrar todas las posiciones donde aparece w en d, o sea, enumerar:

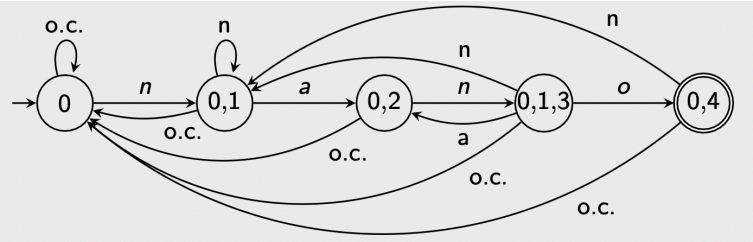
$$\{(i, j) |\, w = d_i d_{i+1} \dots d_j\}$$

Autómata de un patrón

Ejemplo: palabra $w = nano$



Autómata determinizado:



Sea $w = w_1 \dots w_m$ y $\mathcal{A}_w^{det} = (Q^{det}, \Sigma, \delta^{det}, \{0\}, F^{det})$ la determinización de $\mathcal{A}_w$.

Teorema

Para todo $S \in Q^{det}$ y $i \in \{0,1, \dots, m\}$ se cumple que:

$$i \in S \text{ si, y solo si, } w_1 \dots w_i \text{ es un sufijo de } w_1 \dots w_{\max(S)}.$$

Corolarios

- Para todo $S_1, S_2 \in Q^{det}$, si $\max(S_1) = \max(S_2)$ entonces $S_1 = S_2$.
- $\mathcal{A}_w^{det}$ tiene $|w| + 1$ estados y a lo más $\mathcal{O}(|w|^2)$ transiciones.

Por lo tanto, encontrar todos los substrings de w en d toma tiempo $\mathcal{O}(|d| + |w|^2)$

Demostración Teorema

Sea $S \in Q^{det}$ un conjunto de estados cualquiera alcanzable desde $\{0\}$.
Entonces existe una palabra $u = a_1 \dots a_k$ tal que $\hat{\delta}^{det}(\{0\}, u) = S$.
Por la demostración que $\mathcal{L}(\mathcal{A}^{det}) = \mathcal{L}(\mathcal{A})$ para todo NFA $\mathcal{A}$, sabemos que $j \in S$ si, y sollo si, existe una ejecución de $\mathcal{A}_w$ sobre u:

$$0 = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k = j$$

Por la definición de $\mathcal{A}_w$ esta ejecución es de la forma:

$$0 \xrightarrow{a_1} 0 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-j}} 0 \underbrace{\xrightarrow{a_{k-j+1}} 1 \xrightarrow{a_{k-j+2}} 2 \xrightarrow{a_{k-j+3}} \cdots \xrightarrow{a_k} j}_{w_1 \dots w_j}$$

Por lo tanto, $w_1 w_2 \dots w_j$ es sufijo de $a_1 \dots a_k$.

Propiedad

Para toda $u = a_1 \dots a_k$ tal que $\hat{\delta}^{det}(\{0\}, u) = S$, y para todo $j \leq m$:

$$j \in S \text{ si, y solo si, } w_1 \dots w_j \text{ es sufijo de } a_1 \dots a_k$$

Demostración del teorema (⇒)

Como S es alcanzable dese $\{0\}$,
entonces existe $u = a_1 \dots a_k$ tal que $\hat{\delta}^{det}(\{0\}, u) = S$.

Como $\max(S) \in S$, entonces $w_1 \dots w_{\max(S)}$ es sufijo de $a_1 \dots a_k$.
Suponga que $i \in S$. Entonces $w_1 \dots w_i$ es sufijo de $a_1 \dots a_k$.
Como $i \leq \max(S)$ entonces:

$$a_1 a_2 \dots a_{k-\max(S)} \overbrace{a_{k-\max(S)+1} \dots a_{k-i} \underbrace{a_{k-i+1} \dots a_k}_{w_1 \dots w_i}}^{w_1 \dots w_{\max(S)}}$$

Por lo tanto, $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\max(S)}$.

Demostración del teorema (⇐)

Como S es alcanzable desde $\{0\}$,
entonces existe $u = a_1 \dots a_k$ tal que $\hat{\delta}^{det}(\{0\}, u) = S$.

Como $\max(S) \in S$, entonces $w_1 \dots w_{\max(S)}$ es sufijo de $a_1 \dots a_k$.
Suponga que $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\max(S)}$.
Como $w_1 \dots w_i$ es sufijo de $w_1 \dots w_{\max(S)}$ y $w_1 \dots w_{\max(S)}$ es sufijo de u,
entonces $w_1 \dots w_i$ es sufijo de $u = a_1 \dots a_k$.

Por la "Propiedad", concluimos que $i \in S$.

Autómata finito con *k*-lookahead

Sea Σ un alfabeto finito.

Definiciones

Se definen los siguientes conjuntos de palabra:

- $\Sigma_{\blacksquare} = \Sigma^* \times \Sigma^*$
- $\Sigma_{\blacksquare}^k = \{(u, v) \in \Sigma_{\blacksquare} \mid |uv| = k\}$

Notación

En vez de $(u, v) \in \Sigma_{\blacksquare}$, escribiremos $u.v \in \Sigma_{\blacksquare}$.

Definición

Un autómata finito determinista con *k*-lookahead es:

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

- Q es un conjunto finito de estados.
- Σ es el alfabeto de input.
- q_0 es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

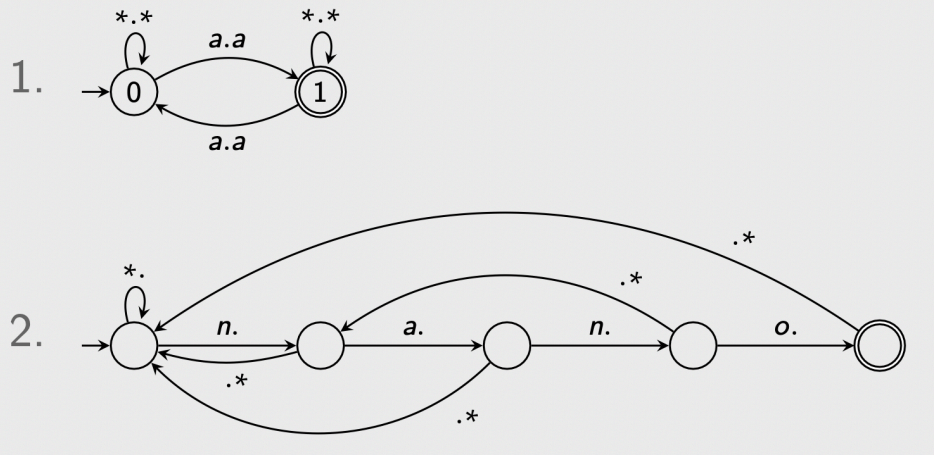
+

- $\delta: Q \times (\Sigma \cup \{\$\})^k \rightharpoonup Q$ es una función parcial, tal que:

$ ≈ End Of String

Para todo $p \in Q$ y $w \in (\Sigma \cup \{\$\})^k$: $|\{u.v \mid \delta(p, u.v) = q \ \ y \ \ uv = w\}| \leq 1$

Algunos ejemplos:



Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA con *k*-lookahead.

Definiciones:

- Un par $(q, w) \in Q \times (\Sigma \cup \{\$\})^*$ es una **configuración** de $\mathcal{A}$.
- Una configuración $(q_0, w\$^k)$ es **inicial**.
- Una configuración $(q, \$^k)$ es **final** si $q \in F$.

El sufijo $\$^k$ nos sirve para marcar el final del input (y simplificar la definición de lookahead al leer el final de la palabra)

Definiciones

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

$$(p_1, w_1) \vdash_{\mathcal{A}} (p_2, w_2)$$

si, y solo si, $\delta(p_1, u.v) = p_2$ y existe $w \in \Sigma^*$ tal que $w_1 = uvw$ y $w_2 = vw$.

- $\mathcal{A}$ **acepta** w si existe una configuración **inicial** $(q_0, w\$^k)$ y una configuración **final** $(q_f, \$^k)$ tal que:

$$(q_0, w\$^k) \vdash_{\mathcal{A}}^* (q_f, \$^k)$$

- Llamaremos un **lazy autómata** a un DFA con 1-lookahead.

Algoritmo de Knuth-Morris-Prat

Sea $w = w_1 \dots w_m$ y $\mathcal{A}_w^{det} = (Q^{det}, \Sigma, \delta^{det}, \{0\}, F^{det})$ la determinización de $\mathcal{A}_w$.

Definición

Para $i \in [0, m]$, sea S_i el **único estado** en Q^{det} tal que $i = \max(S_i)$.

Propiedad 2

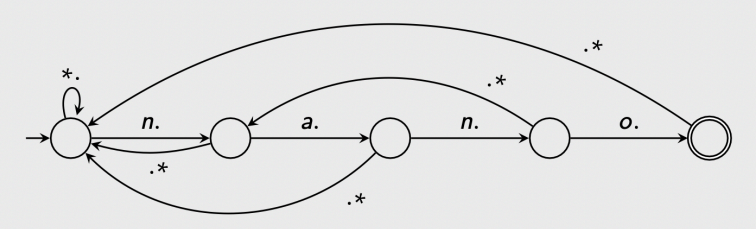
Para todo $a \in \{w_1, .., w_m\}$ y $i \in [0, m-1]$:

1. $S_i - \{i\} \in Q^{det}$.
2. $a = w_{i+1}$, entonces $\delta^{det}(S_i, a) = S_{i+1}$.
3. $a \neq w_{i+1}$, entonces $\delta^{det}(S_i, a) = \delta^{det}(S_i - \{i\})$.

Construcción

Se define el lazy autómata $\mathcal{A}_w^{lazy} = (Q^{det}, \Sigma, \delta^{lazy}, \{0\}, F^{det})$ tal que:

- Para todo $a \neq w_1$: $\delta^{lazy}(\{0\}, a.) = \{0\}$.
- Para todo $a \in \{w_1, \dots, w_m\}$ y $i \in [0, m-1]$:
  - Si $a = w_{i+1}$, entonces $\delta^{lazy}(S_i, a.) = S_{i+1}$.
  - Si $a \neq w_{i+1}$ y $i \neq 0$, entonces $\delta^{lazy}(S_i, .a) = S_i - \{i\}$.



¿Cuántos pasos toma $\mathcal{A}_w^{lazy}$ sobre un documento d?

- Número de pasos que $\mathcal{A}_w^{lazy}$ **consume** letras $= |d|$
- Número de pasos que $\mathcal{A}_w^{lazy}$ **retrocede** $\leq d$
- Número de **pasos totales** de $\mathcal{A}_w^{lazy} \leq 2 \cdot |d|$

Gramáticas libres de contexto

Una **gramática libre de contexto** (CFG) es una tupla:

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

- V es un conjunto finito de **variables** o **no-terminales**.
- Σ es un alfabeto finito (o **terminales**) tal que $\Sigma \cap V = \emptyset$.
- $P \subseteq V \times (V \cup \Sigma)^*$ es un subconjunto finito de **reglas** o **producciones**.
- $S \in V$ es la **variable inicial**.

Ejemplo:

$$\begin{array}{rlll} \mathcal{G}: & A & \rightarrow & 0\ A\ 1 \\ & A & \rightarrow & B \\ & B & \rightarrow & \# \end{array}$$

La gramática se define como $\mathcal{G} = (V, \Sigma, P, S)$ tal que:

- $V = \{ A, B \}$
- $\Sigma = \{ 0, 1, \# \}$
- $P = \{ A \rightarrow 0A1, A \rightarrow B, B \rightarrow \# \}$
- $S = A$

→ Variables en **mayus**
→ Letras en **minus**
→ **Producciones**

Notación:

- Para las **variables** en una gramática usaremos letras mayúsculas:
  $$A, B, C, \ldots$$
- Para los **terminales** en una gramática usaremos letras minúsculas.
  $$a, b, c, \ldots$$
- Para palabras en $(V \cup \Sigma)^*$ usaremos símbolos:
  $$\alpha, \beta, \gamma, \ldots$$
- Para una producción $(A, \alpha) \in P$ la escribimos como:
  $$A \rightarrow \alpha$$

Simplificación

Si tenemos un conjunto de reglas de la forma:

$$\begin{array}{lcl} A & \to & \alpha_1 \\ A & \to & \alpha_2 \\ & \dots & \\ A & \to & \alpha_3 \end{array}$$

entonces escribimos estas reglas **sucintamente** como:

$$A \to \alpha_1 \mid \alpha_2 \mid \alpha_n$$

(recordar que: $\alpha_1, \alpha_2, \dots, \alpha_n \in (\Sigma \cup V)^*$)

Ejemplo anterior:

$$\begin{array}{lcl} \mathcal{G}: \quad A & \to & 0\ A\ 1 \\ A & \to & B \\ B & \to & \# \end{array}$$

Notación sucinta:

$$\begin{array}{lcl} \mathcal{G}: \quad A & \to & 0\ A\ 1 \mid B \\ B & \to & \# \end{array}$$

Producciones

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG.

Definición:

Definimos la relación $\Rightarrow \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ de **producción** tal que:

$$\alpha \cdot A \cdot \beta \Rightarrow \alpha \cdot \gamma \cdot \beta \qquad \text{si, y solo si,} \qquad (A \to \gamma) \in P$$

para todo $A \in V$ y $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$

Si $\alpha A \beta \Rightarrow \alpha \gamma \beta$ entonces decimos que:

- $\alpha A \beta$ **produce** $\alpha \gamma \beta$ o
- $\alpha \gamma \beta$ **es producible** desde $\alpha A \beta$.

$\alpha A \beta \Rightarrow \alpha \gamma \beta$ es reemplazar γ en A en la palabra $\alpha A \beta$.

¿Cuál de las siguientes producciones son correctas?

$$\begin{array}{rlll} \mathcal{G}: & A & \rightarrow & 0\,A\,1 \mid B \\ & B & \rightarrow & \# \end{array}$$

- $A \Rightarrow B$ **SI**
- $00A11 \Rightarrow 000A111$ **SI**
- $000B111 \Rightarrow 000A111$ **NO**
- $0A0A1B \underbrace{A}_{0A1} \Rightarrow 0A0A1B0A1$ **SI**

Definición

Dada dos palabras $\alpha, \beta \in (V \cup \Sigma)^*$ decimos que α **deriva** β:

$$\alpha \overset{*}{\Rightarrow} \beta$$

Si existe $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup \Sigma)^*$ tal que:

$$\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \beta$$

con $\overset{*}{\Rightarrow}$ es la clausura refleja y transitiva de $\Rightarrow$, esto es:

1. $\alpha \overset{*}{\Rightarrow} \alpha$
2. $\alpha \overset{*}{\Rightarrow} \beta$ si, y solo si, existe γ tal que $\alpha \overset{*}{\Rightarrow} \gamma$ y $\gamma \Rightarrow \beta$ para todo $\alpha, \beta \in (V \cup \Sigma)^*$.

Notar que $\Rightarrow$ y $\overset{*}{\Rightarrow}$ son relaciones entre palabras en $(V \cup \Sigma)^*$

¿Cuál de las siguientes derivaciones son correctas?

$$\begin{array}{rlll} \mathcal{G}: & A & \rightarrow & 0\,A\,1 \mid B \\ & B & \rightarrow & \# \end{array}$$

- $A \overset{*}{\Rightarrow} 000A111$ **SI**
- $00A11 \overset{*}{\Rightarrow} 000B111$ **NO**
- $00A11 \overset{*}{\Rightarrow} 000\#111$ **SI**

Definición
El lenguaje de una gramática $\mathcal{G}$ se define como:

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

$\mathcal{L}(\mathcal{G})$ son todas las palabras en Σ^* que se pueden derivar desde S

¿Qué palabras están en $\mathcal{L}(\mathcal{G})$?

$$\begin{array}{llll} \mathcal{G}: & A & \rightarrow & 0\,A\,1 \mid B \\ & B & \rightarrow & \# \end{array}$$

- Como $A \overset{*}{\Rightarrow} 000\#111$, entonces $000\#111 \in \mathcal{L}(\mathcal{G})$.
- En general, uno puede demostrar por inducción que:

$$\mathcal{L}(\mathcal{G}) = \{\, 0^n\#1^n \mid n \geq 0 \,\}$$

¿Qué lenguaje define cada gramática libre de contexto?

1. $\begin{array}{llll} \mathcal{G}: & S & \rightarrow & AS \mid \epsilon \\ & A & \rightarrow & aa \mid ab \mid ba \mid bb \end{array}$

   Palabras de largo par

2. $\begin{array}{llll} \mathcal{G}: & S & \rightarrow & S + S \mid S \times S \mid (S) \mid A \\ & A & \rightarrow & 0 \mid 1 \mid \ldots \mid 9 \end{array}$

   Todas las expresiones aritméticas válidas del 0 al 9

3. $\mathcal{G}: \quad S \rightarrow a\,S\,b \mid S\,S \mid \epsilon$

   Si reemplazo $'a'$ y $'b'$ por $'('$ y $')'$ reespectivamente, el lenguaje corresponde al conjunto de paréntesis bien anidados (es decir, paréntesis puestos de manera correcta)

¿Cuál es una gramática para cada lenguaje?

1. $L_1 = \{\, 0^n1^n \mid n \geq 0 \,\} \cup \{\, 1^n0^n \mid n \geq 0 \,\}$

   1
   $S \rightarrow A \mid B$
   $A \rightarrow 0A1 \mid \epsilon$
   $B \rightarrow 1B0 \mid \epsilon$

2. $L_2 = \{\, w \in \{a, b\}^* \mid w = w^{\text{rev}} \,\}$

   2
   $S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

Definición

Diremos que $L \subseteq \Sigma^*$ es un **lenguaje libre de contexto** ssi existe una gramática libre de contexto $\mathcal{G}$ tal que:

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

Ejemplos:

- $L = \{\, 0^n\#1^n \mid n \geq 0 \,\}$
- $Par = \{\, w \in \{a, b\}^* \mid w \text{ tiene largo par} \,\}$
- $Pal = \{\, w \in \{a, b\}^* \mid w = w^{\text{rev}} \,\}$

Árboles ordenados y etiquetados

Definiciones

El conjunto de **árboles ordenados y etiquetados** (o solo **árboles**) sobre etiquetas Σ y V, se define recursivamente como:

- $t \coloneqq a$ es un árbol para todo $a \in \Sigma$.
- Si $t_1, \ldots, t_k$ son árboles,
  Entonces $t \coloneqq A(t_1, \ldots, t_k)$ es un árbol para todo $A \in V$.

Para un árbol $t \coloneqq A(t_1, \ldots, t_k)$ cualquiera se define:

- $\text{raiz}(t) = A$
- $\text{hijos}(t) = t_1, \ldots, t_k$

Si $t = a$, entonces decimos que t es una hoja, $\text{raiz}(t) = a$ y $\text{hijos}(t) = \epsilon$.

Definiciones

Se define el conjunto de **árboles de derivación** recursivamente como:

- Si $a \in \Sigma$, entonces $t = a$ es un árbol de derivación.
- Si $A \to A_1 \ldots A_k \in P$ y
  $t_1, \ldots, t_k$ son árboles de derivación con $\text{raiz}(t_i) = A_i$ para todo $i \leq k$ entonces
  $t = A(t_1, \ldots, t_k)$ es un árbol de derivación.

Decimos que t es un **árbol de derivación de $\mathcal{G}$** si:

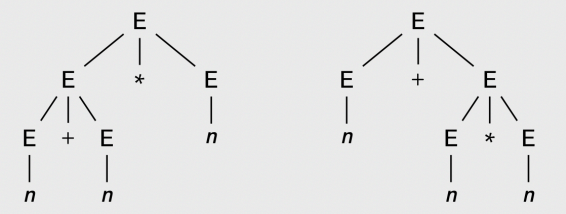
1. t es un árbol de derivación y
2. $\text{raiz}(t) = S$

Los árboles de derivación son todos los árboles que parten desde S

Ejemplo de árbol de derivación

$$\mathcal{G}: \quad E \;\to\; E + E \mid E * E \mid n$$

Algunos **árboles de derivación** para $\mathcal{G}$:



Definiciones

Se define la función **yield** sobre árboles, recursivamente como:

- Si $t = a \in \Sigma$, entonces $\text{yield}(t) = a$.
- Si t no es una hoja e $\text{hijos}(t) = t_1 t_2 \ldots t_k$ entonces:

$$\text{yield}(t) = \text{yield}(t_1) \cdot \text{yield}(t_2) \cdot \ldots \cdot \text{yield}(t_k)$$

Decimos que t es un **árbol de derivación de $\mathcal{G}$ para w** si:

1. t es un árbol de derivación de $\mathcal{G}$ y
2. $\text{yield}(t) = w$.

Las hojas de t forman la palabra w

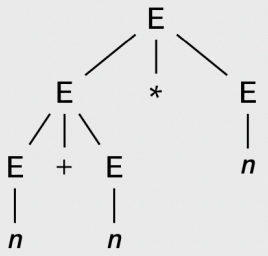
Proposición

$w \in \mathcal{L}(\mathcal{G})$ si, y solo si, existe un árbol de derivación de $\mathcal{G}$ para w.

Un árbol de derivación es la representación gráfica de una derivación

Ejemplo:

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



1) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow n+E*E \Rightarrow n+n*E \Rightarrow n+n*n$
2) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow E+n*E \Rightarrow E+n*n \Rightarrow n+n*n$
3) $E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow E+E*n \Rightarrow E+n*n \Rightarrow n+n*n$
4) $E \Rightarrow E*E \Rightarrow E*n \Rightarrow E+E*n \Rightarrow n+E*n \Rightarrow n+n*n$
5) $E \Rightarrow E*E \Rightarrow E*n \Rightarrow E+E*n \Rightarrow E+n*n \Rightarrow n+n*n$
6) ...

Dado un árbol de derivación, ¿Con cuál nos quedamos?

Definición

- Definimos la **derivación por la izquierda** $\underset{\text{lm}}{\Rightarrow} \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$:

$$w \cdot A \cdot \beta \underset{\text{lm}}{\Rightarrow} w \cdot \gamma \cdot \beta \quad \text{si, y solo si,} \quad A \to \gamma \in P$$

  para todo $A \in V$, $w \in \Sigma^*$ y $\beta, \gamma \in (V \cup \Sigma)^*$.

- Definimos la derivación por la derecha $\underset{\text{rm}}{\Rightarrow} \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$:

$$\alpha \cdot A \cdot w \underset{\text{rm}}{\Rightarrow} \alpha \cdot \gamma \cdot w \quad \text{si, y solo si,} \quad A \to \gamma \in P$$

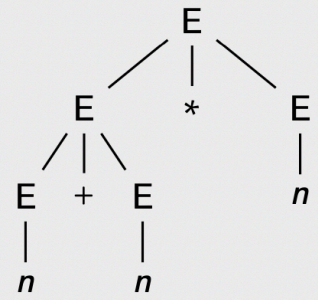
  para todo $A \in V$, $w \in \Sigma^*$ y $\alpha, \gamma \in (V \cup \Sigma)^*$.

Se define $\overset{*}{\underset{\text{lm}}{\Rightarrow}}$ y $\overset{*}{\underset{\text{rm}}{\Rightarrow}}$ como la clausura refleja y transitiva de $\underset{\text{lm}}{\Rightarrow}$ y $\underset{\text{rm}}{\Rightarrow}$, respectivamente.

$\underset{\text{lm}}{\Rightarrow}$ y $\underset{\text{rm}}{\Rightarrow}$ solo reemplaza a la **izquierda** (leftmost) y **derecha** (rightmost)

Ejemplo anterior:

$$E \to E + E \mid E * E \mid n$$



Derivación por la izquierda (lm)

$$E \underset{\text{lm}}{\Rightarrow} E * E \underset{\text{lm}}{\Rightarrow} E + E * E \underset{\text{lm}}{\Rightarrow} n + E * E \underset{\text{lm}}{\Rightarrow} n + n * E \underset{\text{lm}}{\Rightarrow} n + n * n$$

Derivación por la derecha (rm)

$$E \underset{\text{rm}}{\Rightarrow} E * E \underset{\text{rm}}{\Rightarrow} E * n \underset{\text{rm}}{\Rightarrow} E + E * n \underset{\text{rm}}{\Rightarrow} E + n * n \underset{\text{rm}}{\Rightarrow} n + n * n$$

¿Cuál es la relación entre el **tipo de derivación** y el **recorrido del árbol**?

Sabemos que…

- Por cada derivación, existe un único árbol de derivación.
- Por cada árbol de derivación existen **múltiples** posibles derivaciones.

Proposición

Por cada árbol de derivación, existe una **única** derivación por la izquierda y una **única** derivación por la derecha.

Simplificación de gramáticas

Variables inútiles

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG.

Definición

Diremos que una variable $X \in V$ es **útil** si existe una derivación:

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$$

Al contrario, diremos que una variable X es **inútil** si NO es útil.

¿Qué variables son inútiles?

$$\begin{aligned} S &\rightarrow aAa \mid aBC \\ A &\rightarrow aS \mid bD \\ B &\rightarrow aBa \mid b. \\ C &\rightarrow abb \mid DD. \\ D &\rightarrow aDa \end{aligned}$$

D: Una vez obtengo una D, no puedo llegar a nada.

Definición

Para una variable $X \in V$:

1. Decimos que X es **alcanzable** si existe una derivación:

$$S \overset{*}{\Rightarrow} \alpha X \beta$$

2. Decimos que X es **generadora** si existe una derivación:

$$X \overset{*}{\Rightarrow} w$$

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$
output: Conjunto G de variables generadoras

Function `Generadores` $(\mathcal{G})$
- **let** $G_0 := \{X \in V \mid (X \to w) \in P\}$
- **let** $G := \varnothing$
- **while** $G_0 \neq G$ **do**
  - $G := G_0$
  - **foreach** $(X \to \alpha) \in P$ **do**
    - **if** *todas las variables en* α *estan en* G **then**
      - $G_0 := G_0 \cup \{X\}$
- **return** G

Propiedad

Para toda variable $X \in V$:

Existe una regla $X \to \alpha$ tal que todas las variables en α son generadoras
si y solo si, X es **generadora**.

¿Cuáles variables son generadoras?

$S \to aAa \mid aBD$
$A \to aB \mid bD$
$B \to aBa \mid b$
$C \to abb \mid DD$
$D \to aDCa$

$G_0 = \{B, C\}$
$G_1 = \{A\}$
$G_2 = \{S\}$
$\boldsymbol{G} = \{A, B, C, S\}$

input : Gramática $\mathcal{G} = (V, \Sigma, P, S)$
output: Conjunto C de variables alcanzables

Function `alcanzables` $(\mathcal{G})$
- **let** $C_0 := \{S\}$
- **let** $C := \varnothing$
- **while** $C_0 \neq \varnothing$ **do**
  - **take** $Y \in C_0$
  - $C_0 := C_0 - \{Y\}$
  - $C := C \cup \{Y\}$
  - **foreach** $X \in V - C$ *tal que existe una regla* $(Y \to \alpha X \beta) \in P$ **do**
    - $C_0 := C_0 \cup \{X\}$
- **return** C

Propiedad

Para toda variable $X \in V - \{S\}$:

Existe una producción $Y \to \alpha X \beta$ en P tal que $Y \in V$ es alcanzable
si, y solo si, X es **alcanzable**.

¿Cuáles variables son alcanzables?

$S \to aAa$
$A \to aB$
$B \to aBa \mid b$
$C \to abb$

$C_0 = \{S\}$ $C = \emptyset$	$C_0 = \{A\}$ $C = \{S\}$	$C_0 = \{B\}$ $C = \{S, A\}$	$C_0 = \emptyset$ $C = \{S, A, B\}$

Teorema

Sea $\mathcal{G}''$ una gramática creada a partir de $\mathcal{G}$ después de:

- Eliminar todas las variables y reglas **NO** **generadoras**.
- Eliminar todas las variables y reglas **NO** **alcanzables**.

Entonces, $\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G})$ y $\mathcal{G}''$ no contiene variables inútiles.

Demostración

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG.

Sea $\mathcal{G}' = (V', \Sigma, P', S)$ al eliminar las variables **no generadoras** de $\mathcal{G}$:

$$V' = \{ X \in V \mid \exists w.\ X \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \mathrm{w} \}$$
$$P' = \{ X \to \alpha \in P \mid X \in V' \wedge \alpha \in (V' \cup \Sigma)^* \}$$

Sea $\mathcal{G}'' = (V'', \Sigma, P'', S)$ al eliminar las variables **no alcanzables** de $\mathcal{G}'$:

$$V'' = \{ X \in V' \mid \exists \alpha, \beta.\ S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha X \beta \}$$
$$P'' = \{ X \to \alpha \in P' \mid X \in V'' \wedge \alpha \in (V'' \cup \Sigma)^* \}$$

Considere las propiedades de $\mathcal{G}, \mathcal{G}', \mathcal{G}''$.

1. Para todo $\alpha \in (V \cup \Sigma)^*$, si $\alpha \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$ entonces $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.
2. Para todo $\alpha \in (V' \cup \Sigma)^*$, si $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha$ entonces $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha$.
3. Para todo $\alpha \in (V'' \cup \Sigma)^*$, si $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$ entonces $\alpha \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

Demostración

PD: $\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G})$.

Como $V'' \subseteq V$ y $P'' \subseteq P$, entonces $\mathcal{L}(\mathcal{G}'') \subseteq \mathcal{L}(\mathcal{G})$.

PD: $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}'')$.

Sea $w \in \mathcal{L}(\mathcal{G})$ tal que $S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$.

- Por la propiedad 1. tenemos que $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.
- Por la propiedad 2. tenemos que $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

Por lo tanto $w \in \mathcal{L}(\mathcal{G}'')$ y concluimos que $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}'')$.

Demostración

PD: Para todo $X \in V''$, X **es útil** en $\mathcal{G}''$.

Como $X \in V''$, entonces $S \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} \alpha X \beta$ para algún $\alpha, \beta \in (V' \cup \Sigma)^*$.

Por la propiedad 2. se tiene que $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha X \beta$ y $\alpha, \beta \in (V'' \cup \Sigma)^*$.

Como $X \in V'$ y $\alpha, \beta \in (V' \cup \Sigma)^*$, entonces existen u, v, w tal que:

$$\alpha \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} u, \qquad X \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} v, \qquad \beta \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} w$$

Por la propiedad 1. se tiene que: $\alpha \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} u$, $X \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} v$, $\beta \overset{*}{\underset{\mathcal{G}'}{\Rightarrow}} w$.

Por la propiedad 3. se tiene que: $\alpha \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} u$, $X \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} v$, $\beta \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} w$.

Juntando todo $S \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} \alpha X \beta \overset{*}{\underset{\mathcal{G}''}{\Rightarrow}} uvw$ y X es útil en $\mathcal{G}''$.

¿Qué falla al eliminar primero las no alcanzables y después las no generadoras?

Ejemplo:

$$\begin{array}{lrcl} \mathcal{G}: & S & \to & AB \mid b \\ & A & \to & a \\ & B & \to & B \end{array}$$

Al eliminar variables **no alcanzables** en $\mathcal{G}$:

$$\begin{array}{lrcl} \mathcal{G}': & S & \to & AB \mid b \\ & A & \to & a \\ & B & \to & B \end{array}$$

Al eliminar variables **no generadoras** en $\mathcal{G}'$:

$$\begin{array}{lrcl} \mathcal{G}'': & S & \to & b \\ & A & \to & a \end{array}$$

(A es inútil)

Definición

- Decimos que una producción de la forma: $X \to \epsilon$ es **en vacío**.
- Decimos que una producción de la forma: $X \to Y$ es **unitaria**.

Deseamos eliminar este tipo de producciones.

Conclusión

Si $\epsilon \in \mathcal{L}(\mathcal{G})$, entonces
NO se pueden borrar las producciones en vacío sin alterar el lenguaje $\mathcal{G}$.

Desde ahora, supondremos que $\epsilon \notin \mathcal{L}(\mathcal{G})$.

Sea $\mathcal{G} = (V, \Sigma, P, S)$ un CFG tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$.

Observación

Suponga las reglas $X \to Y$ y $Y \to \gamma$ en P.

- Si $\mathcal{G}' = (V, \Sigma, P \cup \{X \to \gamma\}, S)$ $\quad ¿\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})?$ **SI**
- Si $\mathcal{G}'' = (V, \Sigma, P \cup \{X \to \gamma\} - \{X \to Y\}, S)$ $\quad ¿\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G})?$ **NO**

Suponga las reglas $X \to \epsilon$ y $Z \to \alpha X \beta$ en P.

- Si $\mathcal{G}' = (V, \Sigma, P \cup \{Z \to \alpha\beta\}, S)$ $\quad ¿\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})?$ **SI**
- Si $\mathcal{G}'' = (V, \Sigma, P \cup \{Z \to \alpha\beta\} - \{X \to \epsilon\}, S)$ $\quad ¿\mathcal{L}(\mathcal{G}'') = \mathcal{L}(\mathcal{G})?$ **NO**

Clausura de producciones unitarias y en vacío

Sea P^* el **menor conjunto de producciones** que contiene P y **cerrado bajo** las siguientes reglas:

1. Si $X \to Y \in P^*$ y $Y \to \gamma \in P^*$, entonces $X \to \gamma \in P^*$.
2. Si $X \to \epsilon \in P^*$ y $Z \to \alpha X \beta \in P^*$, entonces $Z \to \alpha\beta \in P^*$.

Defina $\mathcal{G}^* = (V, \Sigma, P^*, S)$. Entonces:

- P^* es infinito.
- $\mathcal{L}(\mathcal{G}^*) = \mathcal{L}(\mathcal{G})$

¿Cómo eliminamos las producciones en vacío y unitarias?

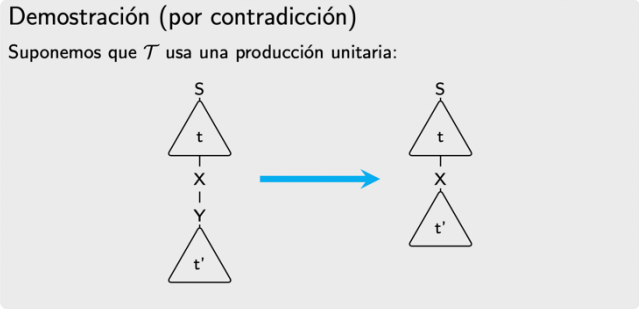
Para cualquier palabra $w \in \mathcal{L}(\mathcal{G}^*)$,
Sea $\mathcal{T}$ un árbol de derivación de w en $\mathcal{G}^*$ de **tamaño mínimo**.

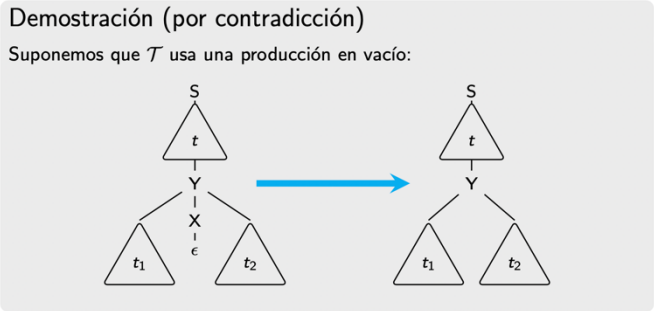
Propiedad 1

El árbol de derivación $\mathcal{T}$ NO usa una **producción unitaria**.

Propiedad 2

El árbol de derivación $\mathcal{T}$ NO usa una producción **en vacío**.





Por la **Propiedad 1** y **Propiedad 2** tenemos que:

Para todo $w \in \mathcal{L}(\mathcal{G}^*)$, existe una derivación de w en $\mathcal{G}$ que NO usa producciones en vacío ni producciones unitarias.

Podemos eliminar las producciones en vacío y unitarias de $\mathcal{G}^*$!

Teorema

Para toda CFG $\mathcal{G}$ tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$, sea:

- $\mathcal{G}^*$ la clausura de producciones unitarias y en vacío.
- $\hat{\mathcal{G}}$ el resultado de remover toda producción unitaria o en vacío de $\mathcal{G}^*$.

Entonces $\mathcal{L}(\hat{\mathcal{G}}) = \mathcal{L}(\mathcal{G})$ y $\hat{\mathcal{G}}$ no tiene producciones unitarias o en vacío.

Para eliminar las producciones en vacío o unitarias de $\mathcal{G}$:

- Construimos $\mathcal{G}^*$ haciendo la clausura de producción unitarias y en vacío,
- Construimos $\hat{\mathcal{G}}$ removiendo todas las producciones unitarias o en vacío de $\mathcal{G}^*$.

Por el resultado anterior sabemos que $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\hat{\mathcal{G}})$.

Forma Normal de Chomsky

Definición

Una gramática $\mathcal{G}$ esta en la **forma normal de Chomsky** (CNF) si todas sus reglas son de la forma:

- $X \to YZ$
- $X \to a$

Toda gramática se puede convertir en CNF

Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG tal que $\epsilon \notin \mathcal{L}(\mathcal{G})$.

- Primero, suponga que $\mathcal{G}$ no contiene las reglas en vacío o unitarias.
- Por lo tanto, todas las reglas en $\mathcal{G}$ son de la forma:
  - $X \to \gamma$ para $|\gamma| \geq 2$
  - $X \to a$

Paso 1: Convertir todas las reglas a la forma:

- $X \to Y_1 Y_2 \dots Y_k$ para $k \geq 2$
- $X \to a$

¿**Cómo**?

- Para cada $a \in \Sigma$, agregar una nueva variable X_a y una regla $X_a \to a$.
- Reemplazar todas las ocurrencias antiguas de a por X_a.

Paso 2: Convertir todas las reglas a la forma:

- $X \to YZ$
- $X \to a$

¿**Cómo**?

Para cada regla p: $X \to Y_1 Y_2 \dots Y_k$ con $k \geq 3$:

- Agregamos una **nueva** variable Z.
- Reemplazamos la regla p por **dos reglas**:

$$X \to Y_1 Z \quad \wedge \quad Z \to Y_2 \dots Y_k$$

Repetimos este paso.

Lema de Bombeo Para lenguajes libres de contexto

Sea $L \subseteq \Sigma^*$. Si L es **libre de contexto**, entonces:

Existe un $N > 0$ tal que
Para toda palabra $z \in L$ con $|z| \geq N$
Existe una descomposición $z = uvwxy$
Con $vx \neq \epsilon$ y $|vwx| \leq N$ tal que
Para todo $i \geq 0$, $u \cdot v^i \cdot w \cdot x^i \cdot y \in L$.

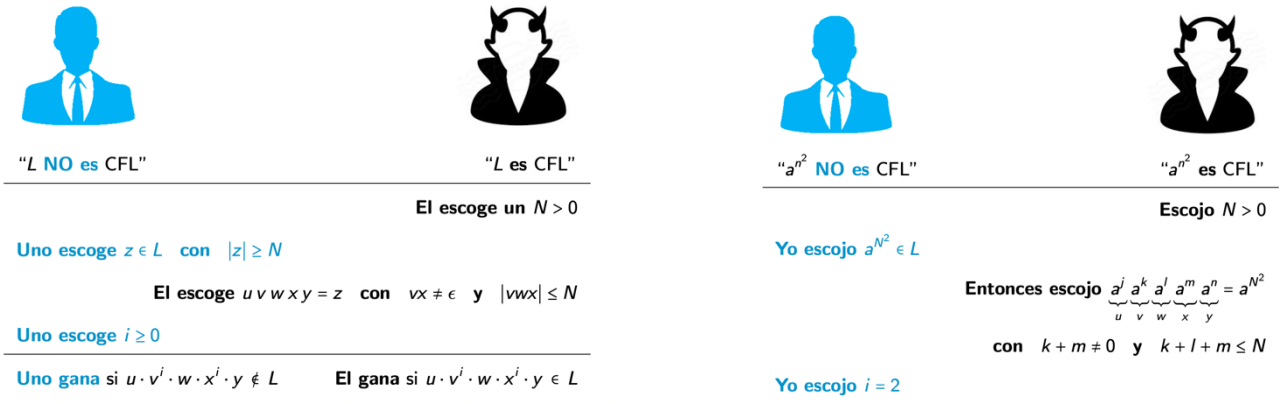
Contra-positivo:

Sea $L \subseteq \Sigma^*$. Si

Para todo $N > 0$ tal que
Existe una palabra $z \in L$ con $|z| \geq N$
Para toda descomposición $z = uvwxy$
Con $vx \neq \epsilon$ y $|vwx| \leq N$ tal que
Existe $i \geq 0$, $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$.

entonces L **NO** es libre de contexto.

Jugando contra un demonio (a^{n^2})



Jugando contra un demonio ($a^n b^n c^n$)