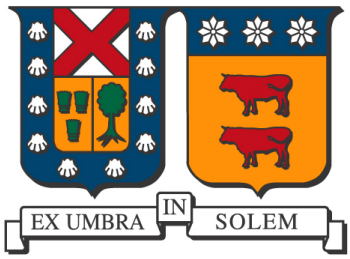


Universidad Técnica Federico Santa María

Informática Teórica

Fernando Iturbe Pemjean

Ultima actualización:
2015/09/28

Índice

1. Evaluaciones

Certámenes:

- C_1: 30 %
- C_2: 35 %

Tareas: 30 %
Ayudantias : 5 %

2. Teoría de lenguajes formales autómatas

2.1. Alfabeto

Conjunto finito de símbolos atómicos. Se anota $\Sigma, \delta, \ldots$

2.2. Palabras (string)

Secuencia finita de símbolos del alfabeto dado. Se anota $\alpha, \beta, \ldots$
El largo de la palabra es su número de símbolos:
$|TALF| = 4$ (también se utiliza lg())

Como estamos hablando de secuencias **el orden importa** y esta permitido que una misma palabra contenga el mismo símbolo:

$|aaa| = 3$

También está permitido tener secuencias de largo cero, lo cual se anota como ϵ.
Una palabra de largo 1 es distinto que una letra

2.3. Operaciones entre palabras

2.3.1. Concatenación

$\alpha \cdot \beta$ es escribir α, luego β. Suele anotarse también $\alpha\beta$.
Si $\alpha = abc$, $\beta = cba$ entonces $\alpha\beta = abccba$ y $\beta\alpha = cbaabc$, en general $\alpha\beta \neq \beta\alpha$.

Notar que $\alpha \cdot \epsilon = \epsilon \cdot \alpha = \alpha$ para todo α.
También: $|\alpha \cdot \beta = |\alpha| + |\beta|$

2.4. Lenguaje sobre el alfabeto

Es un conjunto de palabras sobre Σ.

2.5. Operaciones entre lenguajes

Como los lenguajes son conjuntos tiene sentido aplicar operaciones de estos, tales como unión ($\cap$), intersección ($\cup$), etc.

2.5.1. Concatenación de lenguajes

Se defina la concatenación de lenguajes L_1 y L_2 sobre Σ:

$$L_1 \cdot L_2 = \{\alpha\beta : \alpha \in L_1 \wedge \beta \in L_2\}$$

Tomar en cuenta que en general $L_1 \cdot L_2 \neq L_2 \cdot L_1$
Recordar que el conjunto vacío ϕ es un **lenguaje**, como también ϵ pero $\phi \neq \epsilon$

2.6. Potencias

Sea L un lenguaje sobre Σ, definimos:

- $L^0 = \{\epsilon\}$
- $L^{n+1} = L^n \cdot L$ con $n \geq 0$

Notar que $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$

Definimos la estrella de Kleene:

$$L^* = U_{n \geq 0} L^n$$

(L repetido cero o más veces)

Se usa comúnmente:

$$L^+ = U_{n \geq 1} L^n = L \cdot L^*$$

Notamos antes que si Σ es un alfabeto, $\alpha \in \Sigma$, tenemos ambigüedad. ¿a es un símbolo o una palabra de largo L?
No distinguiremos en aras de simplificar.
De la misma manera, usaremos Σ para el alfabeto y también para el lenguaje de las palabras de largo 1 sobre Σ.
Así Σ^* es una notación cómoda para el conjunto de todas las palabras posibles sobre Σ.

3. Expresiones regulares

Sea Σ un alfabeto, son expresiones regulares sobre Σ que denotan los lenguajes indicados:

- Φ es una expresión regular, denota el lenguaje vacío (Φ).
- ϵ es una expresión regular que denota $\{\epsilon\}$.
- Para todo $a \in \Sigma$: a es una expresión regular que denota $\{a\}$.

Si R y S son expresiones regulares que denotan los lenguajes L_R y L_S respectivamente, entonces:

- $(R|S)$ es una expresión regular que denota $L_R \cup L_S$
- $R \cdot S$ es una expresión regular que denota $L_R \cdot L_S$
- R^* es una expresión regular que denota $L_R^{/ast}$

Acotación

- Se usará comúnmente (R^+) PARA REFERIRNOS A $((L**) \cdot L_R)$
- Usaremos ocasionalmente R^n para potencias fijas en vez de escribir nR_s
- Omitiremos () y $\cdot$ con las reglas como si | es + y $\cdot$ multiplicación

Informalmente:

- $R|S$: RoS una de las dos alternativas
- $R \cdot S$: R, seguido de S
- R^*: 0 o más R
- R^+: 1 o más R

3.0.1. Ejemplos

Con $\Sigma = \{a, b.c\}$

- Palabras que comienzan con a y terminan en c

$$a(a|b|c)^*c$$

- Palabras de largo 2

$$(a|b|c)(a|b|c) = (a|b|c)^2 = aa|ab|ac|ba|bb|bc|ca|cb|cc$$

4. Autómatas finitos deterministas(DFA)

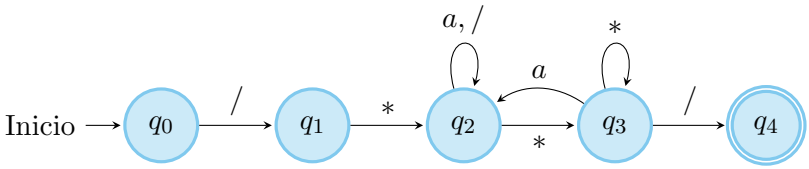
Ejemplo: Comentarios $/*\cdots*/$ en C
Usaremos $\Sigma = \{/, *, a\}$ donde a es cualquier carácter que no sea $/, *$
(agregar grafo)
Formalmente:
Un autómata finito determinista (DFA):

$$M = (Q, \Sigma, \delta, q_0, F)$$

consta de :

- Q: Conjunto finito de estados
- Σ Alfabeto de entrada
- $q_0 \in Q$: Estado inicial
- $F \subseteq Q$: Conjunto de estados finales
- $Q \times \Sigma \to Q$ función de transición

4.1. Notación Gráfica



Formalmente, para comentarios:

$$M = (Q, \Sigma, \delta, q_0, F)$$

con:

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_m\}$
- $\Sigma = \{a, /, *\}$
- $F = \{q_4\}$
- δ

	α	/	$*$
q_0	q_m	q_1	q_m
q_1	q_m	q_m	q_2
q_2	q_2	q_2	q_3
q_3	q_2	q_4	q_3
q_4	q_m	q_m	q_m
q_m	q_m	q_m	q_m

4.1.1. Función de transición extendida

$\hat{\delta} : Q \times \Sigma^* \to Q$ para el DFA $M = (Q, \Sigma, \delta, q_0, F)$ se define mediante:

- $\hat{\delta}(q, \epsilon) = q$ para todo $q \in Q$
- $\hat{\delta}(q, \alpha a) = \delta(\hat{\delta}(q, \epsilon), a)$ para todo $q \in Q$, $\alpha \in \Sigma^*$

4.1.2. Lenguaje aceptado por M

$$L(M) = \{\sigma \in \Sigma^* : \hat{\delta}(q_0, \sigma) \in F\}$$

Comúnmente omitimos el ˆ en $\hat{\delta}$, queda claro de cuál hablamos (o da lo mismo).

4.2. Descripción Instantanea

Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA con $Q \cap \Sigma = \phi$. Una descripción instantánea de M es un string en $\Sigma^* Q \Sigma^*$
Idea $alphaq\beta$ representa la situación en que M procesa $\alpha\beta$; ya leyó *alpha*, está en el estado q y le falta procesar β. Nótese que esta situación no tiene porqué ser posible.

4.3. Relación de transición

Se define la relación de transición de M entre IDs mediante:

$$\alpha q \alpha\beta \vdash_M \alpha a p \beta$$

Si $\delta(q, a) = p$
Generalmente se omite el subíndice en $\vdash_M$, si se subentiende M.
Usando lo anterior:

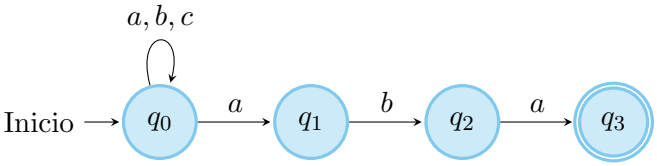
4.3.1. Lenguaje aceptado

$$L(M) = \{\sigma : q_0\sigma \vdash_M \sigma q_+ \wedge q_+ \in F\}$$

5. Autómatas finitos no deterministicos (NFA)

En un DFA las movidas están completamente determinadas, un NFA puede *adivinar* el camino correcto.

Por ejemplo, palabras que terminan aba:



Daremos además al NFA la posibilidad de cambiar de estado sin consumir nada (transición espontantea con ϵ)

Formalmente un DFA con $M = (Q, \Sigma, \delta, q_0, F)$ consta de :

- Q: Conjunto finito de estados
- Σ Alfabeto
- $q_0 \in Q$: Estado inicial
- $\delta : Q \times (\{\epsilon\} \cup \Sigma) \to 2^Q$

5.1. IDs de M

Palabras en $\Sigma^* Q \Sigma^*$ y $\vdash$ por: Si $p \in \delta(q, \epsilon)$:

$$\alpha q \beta \vdash_M \alpha p \beta$$

Si $p \in \delta(q, a)$:

$$\alpha q a \beta \vdash_M \alpha a p \beta$$

Con esto el lenguaje aceptado por M es:

$$L(M) = \{\sigma : q_0 \sigma \vdash_M^* \sigma q_+ \wedge q_+ \in F\}$$

Hasta el momento tenemos tres formas: Expresiones regulares, DFA y NFA:

6. Relación entre expresiones regulares, DFA y NFA

Demostraremos que las tres representan los mismos lenguajes:

$$R.E \to NFA \to DFA \to R.E$$

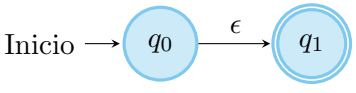
Nótese que $R.E \to NFA \to DFA$ tiene importancia práctica.

6.1. R.E a NFA

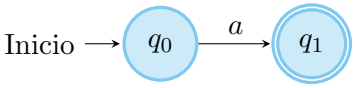
Construimos un NFA siguiendo la definición de la R.E. Los NFA arciales tiene siempre un estado inicial y uno final y nunca pueden *devolverse* al llegar al estado final.

- ϕ es aceptado por

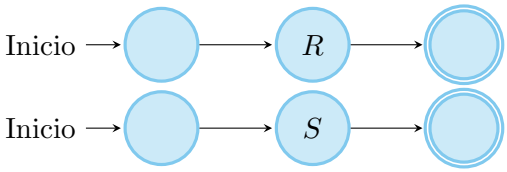
- ϵ es aceptado por



- $a \in \Sigma$ es aceptado por

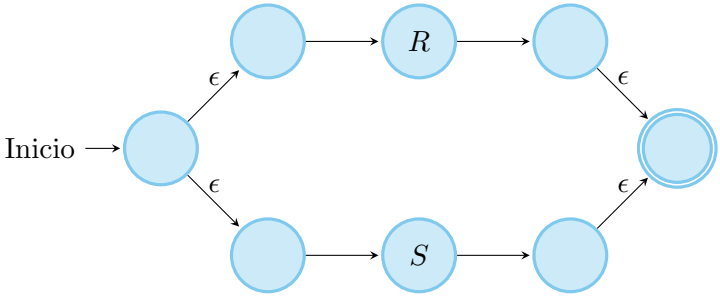


Si R, S son expresiones regulares aceptadas respectivamente por:

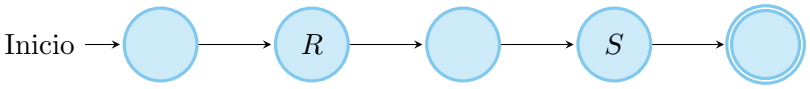


entonces:

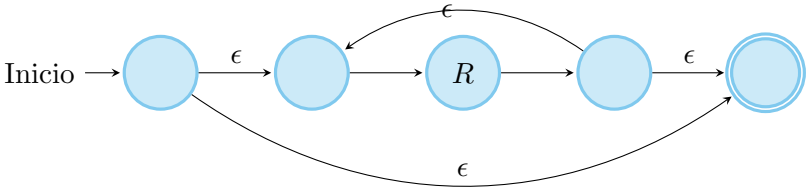
- $(R|S)$ es aceptado por



- $(R \cdot S)$ es aceptado por

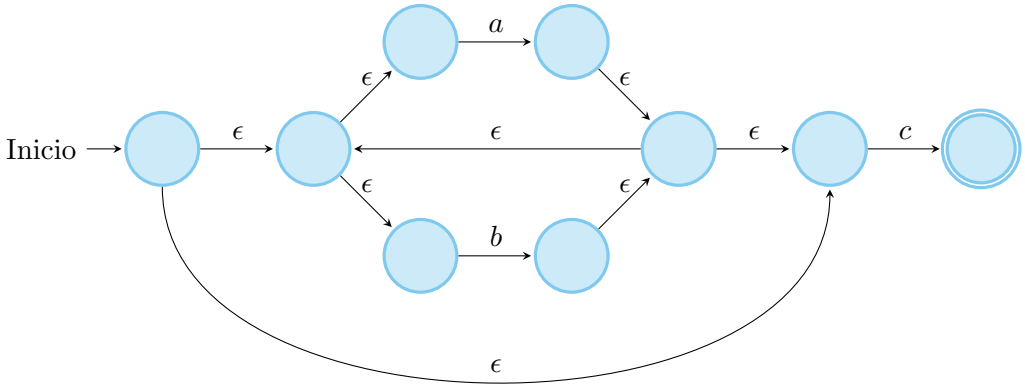


- (R^*) es aceptado por



6.1.1. Ejemplo

$(a|b)^*c$



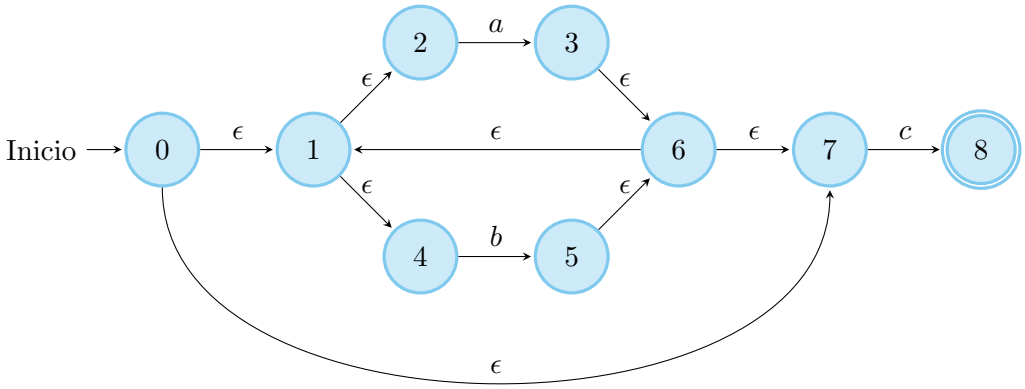
6.2. NFA a DFA

La función de transición extendida de M es $\delta^*(A, x)$ con A un **conjunto de estados** tal que $x \in \Sigma \cup \{\epsilon\}$ y entrega los estados en que M puede estar:

$$\delta^*(A, x) = \{q : p \in A \wedge px \vdash_M xq\}$$

A lo que vamos es construir un DFA cuyos estados son **Conjuntos de estados** del DFA. El DFA resultante en principio tiene $2^{|Q|}$, nos ocuparemos de construir solo aquellos de interés.

6.2.1. Ejemplo



Si tenemos en el conjunto de estados $1, 2, 4$ al consumir a podemos estar en los estados $\{1, 2, 3, 4, 6, 7\}$.

6.2.2. Función e-clousure

Definimos la función ϵ-clousure(A) para el conjunto de estados A como el conjunto de estados alcanzados desde estados en A consumiendo ϵ.

Por ejemplo: ϵ-closure$\{1, 5\} = \{1, 2, 4, 5, 6, 7\}$

- ϵ-closure(A) contiene a A.
- Si ϵ-closure(A) contiene q, contiene todos los estados alcanzables vía ϵ desde q.

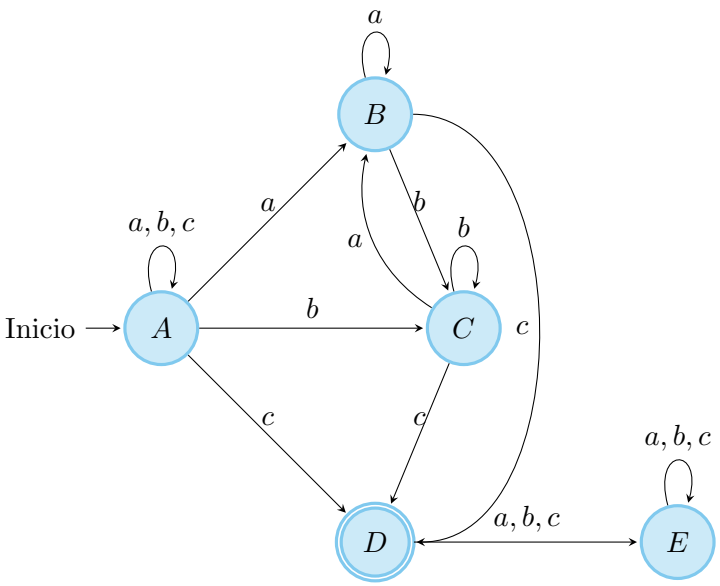
Para construir el DFA equivalente:

- Estado inicial es el conjunto de estados ϵ-closure(q_0)
- Vemos donde nos lleva cada símbolo desde los estados de cada nuevo conjunto, aplicar ϵ-closure
- Cuando dejan de aparecer nuevos conjuntos, estamos listos.
- Estados Finales: Contienen estados finales del NFA

En en el ejemplo: ϵ-closure$(\{0\}) = \{0, 1, 2, 4, 7\}$, lo nombraremos A, entonces $A = \{0, 1, 2, 4, 7\}$
con a: 3, entonces ϵ-closure$(\{3\}) = \{1, 2, 3, 4, 6, 7\}$, lo nombraremos B
Ahora con $b : \{5\}$, entonces ϵ-closure$(\{3\}) = \{1, 2, 4, 5, 6, 7\}$, lo nombraremos C
Con $c : \{8\}$, entonces ϵ-closure$(\{8\}) = \{8\}$, lo nombraremos D
Ahora con $B = \{1, 2, 3, 4, 6, 7\}$
con a: $3 \to B$
Ahora con $b : \{5\} \to C$
Con $c : \{8\} \to D$
Ahora con $C = \{1, 2, 3, 4, 5, 6, 7\}$
con a: $3 \to B$
Ahora con $b : \{5\} \to C$
Con $c : \{8\} \to D$
Ahora con $D = \{8\}$
con $a : \phi \to \epsilon$-closure$(\{8\}) = \phi$, lo nombraremos E Con lo cual tenemos la tabla:

	Conjunto	a	b	c
A	$\{0, 1, 2, 4, 7\}$	B	C	D
B	$\{1, 2, 3, 4, 6, 7\}$	B	C	D
C	$\{1, 2, 4, 5, 6, 7\}$	B	C	D
D	$\{8\}$	E	E	E
E	ϕ	E	E	E

Con lo cual nos queda el siguiente DFA:



6.3. Expresiones Regulares Rij

$R_{ij}^{(k)}$ denota las palabras que llevan a M del estado i al estado j, visitando sólo estados en $1, \ldots, k$.
Notar:

$R_{ij}^{(0)}$ va de i a j directamente (no se permiten estados visitados entre medio), esto es fácil de determinar.
$R_{ij}^{(n)}$ No pone restricciones a los estados visitados.
La unión de $R_{q_0 q_f}^{(n)}$ para $q_f \in F$ es la RE de $L(M)$.
Para ir de i a j, pasando a lo más por k tenemos dos opciones:

1. Nunca visitar k: $R_{ij}^{(k-1)}$
2. Visitar k al menos una vez:
   - Primera vez que llego a k: $R_{ik}^{(k-1)}$
   - Voy de k a k: $R_{kk}^{(k-1)}$
   - Última vez de k a j: $R_{kj}^{(k-1)}$

Es decir:

$$R_{ik}^{(k-1)}(R_{ij}^{(k-1)})^* R_{kj}^{(k-1)}$$

$$R_{ij}^{(k)} = R_{ik}^{(k-1)} | R_{ij}^{(k-1)}(R_{ij}^{(k-1)})^* R_{kj}^{(k-1)}$$

6.3.1. Resumen

Se muestre que las expresiones regulares, NFA, DFA representan la misma colección de lenguajes, la demostración de cada paso demuestra que es efectiva (dado un lenguaje descrito por cualquiera de los tres, podemos conlcluir explícitamente cualquiera de las otras dos), a estos lenguajes se les llama **Lenguajes Regulares**.

6.4. Lenguajes Regulares

6.4.1. Lema de bombeo

Sea L un lenguaje regular, entonces hay una constante $N > 0$ tal que si $\sigma \in L, |\sigma| \geq N$, entonces se puede escribir:

$$\sigma = \alpha\beta\gamma$$

donde $|\alpha\beta| \leq N$, $\beta \neq \epsilon$ u para todo $k \geq 0$ $\alpha\beta^k\gamma \in L$.
Demostración:
Sea $L = \mathbb{L}(M)$, donde M es el DFA $M = (Q, \Sigma, \delta, q_0, F)$, entonces sea $N = |Q|$, consideraremos:

$$\sigma = a_1, a_2, \ldots, a_\Omega$$

con $a_i \in \Sigma$, $l \geq N$ y $\sigma \in L$
Sabemos que M acepta σ, por lo que hay una secuencia de estados $p_0, p_1, p_2, \ldots, p_l$

$$a_1 \ldots a_N \ldots a_{N+1} \ldots a_l$$

$$p_0 \ldots p_1 \ldots p_N \ldots p_{n+1} \ldots p_l$$

los cuales son $N + 1$ estados, ademas tenemos que $p_0 = q_0$ y $p_l \in F$.
Por el principio del palomar, se repite un estado, digamos:

$$p_i = p_j$$

con $i < j$
Con lo cual: $\alpha = a_1 \ldots a_i$

$\beta = a_{a+1} \dots a_j \notin \epsilon$
$\gamma = a_{j+1} \dots a_l$
Sabemos que $j \leq N$, $|\alpha\beta \leq N$, si $i < j$ entonces $\beta \neq \epsilon$, por lo tanto $\alpha\beta^k\gamma \in L$ para todo k, es decir $\alpha\beta^*\gamma$.
Podemos usar esto para demostrar que hay lenguajes que no son regulares mediante contradicción.

6.4.2. Ejemplo

Teorema: El lenguaje $L = a^n b^n : n \geq 0$ no es regular
Demostración: Por contradicción: Supongamos que L es regular, entonces cumple el lema de bombeo, sea N k constante del lema, consideraremos:

$$\sigma = a^N b^N \in L, |\sigma| = 2N \geq N$$

Por lo tanto hay α, β, γ tales que:

$$\alpha\beta\gamma = \sigma, |\alpha\beta| \leq N, \beta \neq \epsilon$$

$$\alpha\beta^k\gamma \in L$$

para todo $k \geq 0$.
Pero como $|\alpha\beta| \leq N$, α y β son solo a, si tomamos $k = 0$, las $|B|$ a faltan y $\alpha\gamma \notin L$, entonces llegamos a una contradicción.
Por lo tanto el lenguaje no es regular.
Otro ejemplo
Sea $L = a^{n^2} : n \geq 0$ no es regular.
Suponemos L regular cumple el Lema de Bombeo.
Sea N la constante del Lema de Bombeo, sea $\sigma = a^{N^2}$, $|\sigma| = N^2 \geq N$.
Por el teorema de bombeo hay α, β, γ con $|\alpha\beta| \leq N, \beta \neq \epsilon$ tal que:

$$\sigma = \alpha\beta\gamma$$

para todo $k \geq 0, \alpha\beta^k\gamma \in L$. Sólo interesan los largos: $|\beta| = u$, sabemos que $\beta \neq \epsilon \rightarrow 1 \leq u \leq N$.
El largo de $|\alpha\beta^k\gamma| = N^2 + (k-1)u$
Elegimos $k = 2$, queremos demostrar que no podrá ser un cuadrado perfecto, por lo tanto:

$$|\alpha\beta^2\gamma| = N^2 + u$$

$$N^2 + 1 \leq N^2 + u \leq N^2 + N$$

tenemos que $N^2 < N^2 + 1$ Y $N^2 + N < N^2 + 2N + 1$, por lo tanto:

$$N^2 < N^2 + u < (N+1)^2$$

Por lo tanto $N^2 + u$ no es un cuadrado perfecto, lo cual es una contradicción.

6.4.3. Ejercicios

1. $a^{n^3} : n \in \mathbb{N}_0$
2. $a^{n!} : n \in \mathbb{N}_0$
3. $a^n b^m c^{n+m}$

6.4.4. Propiedades de Clausura

Teorema: Los lenguajes regulares son cerrados respecto de concatenación, unión, *
Teorema: Los lenguajes regulares son cerrados respecto de complemente.
Demostración Sea L regular y $L = L(M)$ para el DFA $M = (Q, \Sigma, \delta, q_0, F)$. El DFA $M' = (Q, \Sigma, \delta, q_0, Q\ F)$ acepta $\overline{L}$, donde $\overline{L} = \Sigma^*\ L$.

Teorema: Los lenguajes regulares son cerrados respecto intersección.
Demostración: Por leyes de Morgan: $L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$

6.5. Homomorfirsmo

Es una función en $h : \Sigma^* \to \Delta^*$ tal que:

$$h(\alpha \cdot \beta) = h(\alpha) \cdot h(\beta)$$

Claramente es $h(\epsilon) = \epsilon$ y h quede totalmente definida por los valores $h(a)$ para $a \in \Sigma$.

6.6. Substitución

S se le asigna a cada símbolo $a \in \Sigma$, un lenguaje $S_a \in \Delta^*$.
Aplicar S_a a un lenguaje $L \subseteq \Sigma^*$ corresponde a la operación:

$$S(L)) = \{\alpha_1, \alpha_2, \ldots, \alpha_n : a_1, a_2, \ldots, a_n \in L \wedge \alpha_1 \in S_{a_1}, \alpha_2 \in S_{a_2}, \ldots, \alpha_n \in S_{a_n}\}$$

La idea es tomar alguna palabra σ en L, para cada símbolo σ elegir una palabra $\alpha_i \in S_{a_i}$.

6.6.1. Ejemplo

Sea$\Sigma = \{a, e, i, o, u\}$, $L = \{aeio, ui, eo\}$ y $\Delta = \{0, 1, 2\}$.
Entonces:

- $S_a = L((0|1)^*$
- $S_e = n\{\epsilon\}$
- $S_i = L(1^+2)$
- $S_o = \{1^n 2^n : n \geq 0\}$
- $S_u = \Phi$

O sea:
$a \to 1$
$e \to \epsilon$
$i \to 112$
$o \to 111222$
Entonces: $1112111222 \in S(L)$
Teorema: Los lenguajes regulares son cerrados respecto de substitución.
Demostración
Sea L un lenguaje regular sobre Σ y sean S_a lenguajes regulares sobre Δ para todo $a \in \Sigma$. Como L es regular, podemos construir una expresión regular que lo denota. De la misma forma hay una expresión regular para cada S_a. Substituyendo los S_a por a en la expresión regular para L se obtiene una expresión regular para $S(L)$. $\square$.
Ejemplo

$$L = L((a|u)^*|u^+i)$$

- $S_a = L((0|1)^*)$
- $S_u = L(02|1)$
- $S_i = L(011111)$

Entonces

$$S(L) = L((0|1)^*|(02|1)^*|(02|1)^+011111)$$

Teorema: Los lenguajes regulares son regulares respecto de homomorfismos.
Demostración: Un Homomorfismo es una substitución en la cual $S(a)$ es una única palabra. □

6.7. Homomorfismo

Sea $L \subseteq \Sigma^*$ un lenguaje, $h : \Sigma^* \to \Delta^*$ un homomorfismo.
Definimos:

$$h(L) = \{h(\sigma) : \sigma \in L\}$$

6.7.1. Homomorfismo inverso

Sea $B \subseteq \Delta^*$ un lenguaje, $h : \Sigma^* \to \Delta^*$ un homomorfismo.
Se define:

$$h^{-1}(B) = \{\sigma : h(\sigma) \in B\}$$

Teorema: Los lenguajes regulares son cerrados respecto de homomorfismos inversos.
Demostración: Sea L un lenguaje regular sobre Δ, $n : \Sigma^* \to \Delta^*$ un homomorfismo.
Sea $L = L(M)$, donde M es el DFA:

$$M = (Q, \Delta, \delta, q_0, F)$$

Construimos el DFA M':

$$M' = (Q, \Sigma, \delta', q_0, F)$$

tal que $h^{-1} = L(M')$.
Acá $\delta'(q, a) = \delta(q, h(a))$
δ' es una función, M' es un DFA y $\delta'(\sigma) = \delta(h(\sigma))$.
Si $h(\sigma) \in L, \sigma \in h^-1(L)$.
□

7. Gramática

Una gramática $G = (N, \Sigma, P, S)$ consta de:

- N: Alfabeto de no-terminales
- Σ: Alfabeto de terminales Donde $N \cap \Sigma = \phi$
- $V = N \cup \Sigma$ se le llama el vocabulario de G.
- P: Conjunto de Producciones:

  $\alpha \to \beta$donde:

  $\alpha \in V^* N V^*$(α contiene al menos un no-terminal)

  $\beta \in V^*$
- $S \in N$: Símbolo de partida

7.1. Relación de Derivación

Sea $G = (N, \Sigma, P, S)$ una gramática, se define la relación derivación en G sobre V^* mediante:

$$\gamma \implies \delta$$

Si podemos escribir:

$$\gamma = \gamma' \alpha \gamma''$$

$$\delta = \gamma' \beta \gamma''$$

donde $\alpha \to \beta \in P$, es decir γ contiene el lado izquierdo de una producción $\alpha \to \beta$, se obtiene δ reemplazando α por el lado derecho respectivo β.
De acá resulta la relación $\implies^*$ en V^*.
En resumen, $\gamma \implies^* \delta$ si puedo ir de γ a δ en cero o más pasos de $\implies$.

7.1.1. Formas sentenciales

Dada la gramática $G = (N, \Sigma, P, S)$ con vocabulario $V = N \cup \Sigma$, se llaman formas sentenciales de G a las palabras en el conjunto:

$$\{\sigma : S \implies^* \sigma\}$$

El lenguaje generado por G es:

$$L(G) = \{\sigma : S \implies^* \sigma \wedge \sigma \in \Sigma^*\}$$

Idea: $G = (N, \Sigma, P, S)$ es un conjunto de recetas (P) para construir palabras en $\Sigma(L(G))$ partiendo de S.
Ejemplo $G = (N, \Sigma, P, S)$ con

- $N = \{S\}$
- $\Sigma = \{a, b\}$
- $P = \{S \to aSb, S \to \epsilon\}$
- $S = S_0$

Por ejemplo $S \implies \epsilon$, tambien:

$$S \implies aSb$$

$$S \implies aaSbb$$

$$S \implies aabb$$

O sea:

$$\epsilon \in L(G)$$

$$aabb \in L(G)$$

En realidad, no es difícil demostrar que:

$$L(G) = \{a^n b^n : n \geq 0\}$$

7.2. Utilidad práctica

Notación EBNF (y afines) para describir lenguajes de programación.
No terminales: ¡ palabras descriptivas ¿
Terminales: El resto

- Se usa ::= en vez de $\rightarrow$ para indicar producciones.
- Se usa {} para indicar repetición (*)
- Se usa | para alternancia

Por ejemplo
<archivo> ::= { <declaración> | <definición> };
<declaración> ::= <tipo><identificador>;
<definición> ::= <tipo><identificador><valor>;
⋮
<función> ::= <encabezado><cuerpo>;
<encabezado> ::= <tipo><identificador><lista argumentos>;
<cuerpo> ::= ...

7.2.1. Importancia

- Formalismo para describir lenguajes de programación (parcialmente).
- Tipos especiales de gramáticas pueden transformarse a mano o automáticamente en programas que reconstruyen la derivación.

7.3. Clasificación de Gramáticas (Jerarquía de Chomsky)

Sea la gramática $G = (N, \Sigma, P, S)$:

- Toda gramática es de tipo 0 (innestricta)
- Una gramática es de tipo 1 (sensible al contexto si todas sus producciones $\alpha \rightarrow \beta$ cumplen $|\alpha| \leq |\beta|$ (En particular $\beta \neq \epsilon$)

- Una gramática es de tipo 2 (de contexto libre) si todas sus producciones son de la forma:

  $A \to \alpha$

  con $A \in N$ y $\alpha \in V^+$

- Una gramática es de tipo 3 (regular) si todas sus producciones son de la forma:

  $A \to \alpha B$ con $A, B \in N, \alpha \in \Sigma^*$

  $A \to \beta$ con $A \in N, \beta \in \Sigma^*$

- Un lenguaje es del tipo correspondiente si es generado por una gramática de ese tipo.

Notar que:

Tipo 3 $\subset$ Tipo 2 $\subset$ Tipo1 $\subset$ Tipo 0

En gramáticas; veremos lo mismo en lenguajes.

7.4. Convención General

Escribiremos la gramática $G = (N, \Sigma, P, S)$ con las siguientes:

- $N = \{A, B, \dots, S, \dots\}$ (letras mayusculas)
- $\Sigma = \{a, b, \dots\}$ (letras minúsculas y ocasionalmente otros símbolos)

S : generalmente se llama S de todas formas, el no terminal a la izquierda de la primera producción listada.
Para abreviar $A \to \alpha_1$, $A \to \alpha_2$, ..., escribiremos $A \to \alpha_1 | \alpha_2 | \dots$.

7.4.1. Ejemplo de una gramática sensible al contexto (CSG)

$S \to abc_{(1)} | ABC_{(2)}$(Inicio)

$A \to AaM_{(3)} | aF_{(4)}$(Mensaje (continuar/fin))

$Mb \to bM_{(5)}$(Mensaje viaja)

$MB \to bBM_{(6)}$(Recibido en B)

$MC \to Cc_{(7)}$(Recibido en C)

$Fb \to bF_{(8)}$(Mensaje viaja)

$FB \to bF_{(9)}$(Recibido en B)

$FC \to cc_{(10)}$(Recibido en C)

Esta gramática es tipo 1 (sensible al contexto), ya que todas las producciones cumplen $\alpha \to \beta$ con $|\alpha| \leq |\beta|$; no es tipo 2 (contexto libre) por la producción. $Mb \to bM$ (lado izquierdo no es sólo un no-terminal).
Algunas derivaciones:

$S \implies abc(1)$

$S \implies ABC(2)$

$S \implies aAMBC(3)$

$S \implies aAbBMC(6)$

$S \implies aAbBCC(7)$

$S \implies aaFbBCc(4)$

$S \implies aabFbBCc(8)$

$S \implies aabbFCc(9)$

$S \implies aabbccc(10)$

Se ve que:

- Cada vez que se aplica (3) o (4) el numero de a aumenta en 1.
- (5) y (8) no cambian el número de terminales.
- (3), (4), (6), (7) agregan un terminal cada uno, (10) agrega 2.

Por lo tanto:

$$L(G) = \{abc\} \cup \{a^n b^b c^{n+1} : n \geq 1\}$$

7.4.2. Nombre de las gramáticas

- Regular: Las gramáticas regulares generan lenguajes regulares.
- Sensible al contexto: Una definición alternativa son producciones de la forma:

  $$\alpha A \beta \to \alpha \gamma \beta$$

  con $\alpha, \beta \in V^*$, $\gamma \in V^+$.
  Es decir podemos reemplazar A por γ si aparece en el contexto α, β.
- Contexto libre: Puedo reemplazar A por γ sin importar el contexto.

  $$\alpha \to \gamma$$

7.5. Gramáticas regulares v/s autómatas finitos

7.5.1. Idea

En la derivación en una gramática regular **siempre** está el no-terminal al final $\implies$ resume toda la información sobre lo generado antes.
Podemos construir un NFA usando los no terminales como estados.
Al revés, dado un DFA, podemos usar sus estados como no-terminales en una gramática regular.

$S \to abcD$

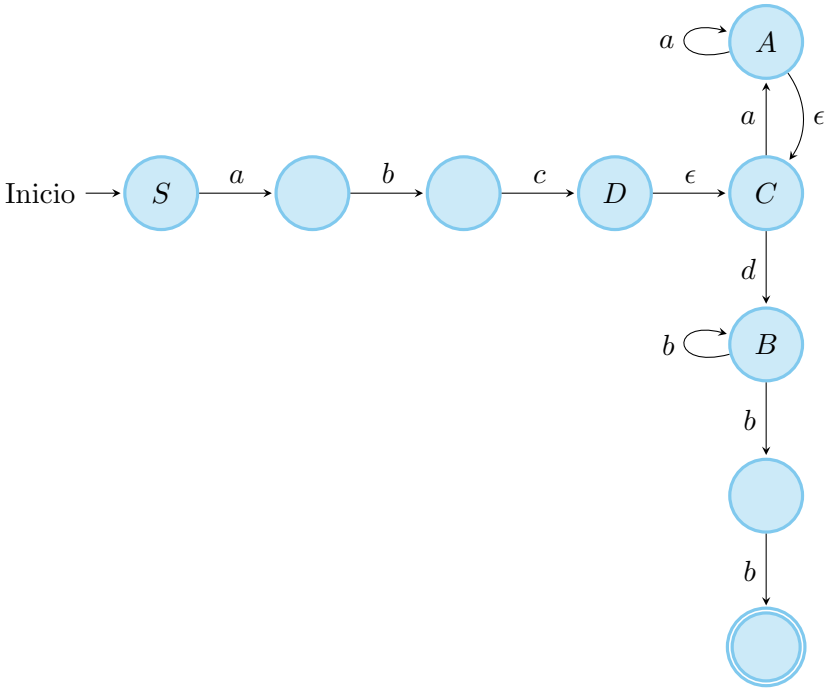
$D \to C$

$C \to dB|aA$

$A \to aA|C$

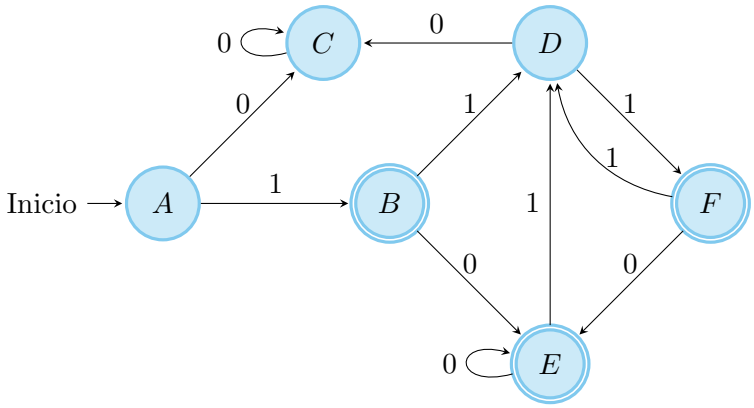
$B \to bB|bb$

Siguiendo la idea, los estados con nombre representan los no-terminales; agregamos estados anónimos segun necesidad. Se inicia con S, estado inicial:



Hay una correspondencia $1-1$ entre derivación en la gramática y computaciones del NFA que aceptan.

7.5.2. Ejemplo



Estrategia
Estado final → producción sin no-terminal
Estado inicial → símbolo de partida.
O sea:

$A \to 0C|1B|1$

$B \to 0E|1D|0$

$C \to 0C|1C$

$D \to 0C|1F|1$

$E \to 0E|1D|0$

$F \to 0E|1D|0$

Nuevamente hay una correspondencia $1-1$ entre computaciones que acepta del DFA y derivaciones.

7.5.3. Esquema

Por lo tanto tenemos que:

$$NFA \to DFA \to GR \to NFA$$

7.6. Gramáticas y lenguajes de contexto libre

Una CFG consta de producciones de K formas:

$$A \to \alpha$$

donde $A \in N$ y $\alpha \in V^+$.

7.6.1. Ejemplo (Gramática regalona)

$$E \to E + T(1)$$

$$E \to T(2)$$

$$T \to T * F(3)$$

$$T \to F(4)$$

$$T \to (E)(5)$$

$$F \to a(6)$$

Entonces:

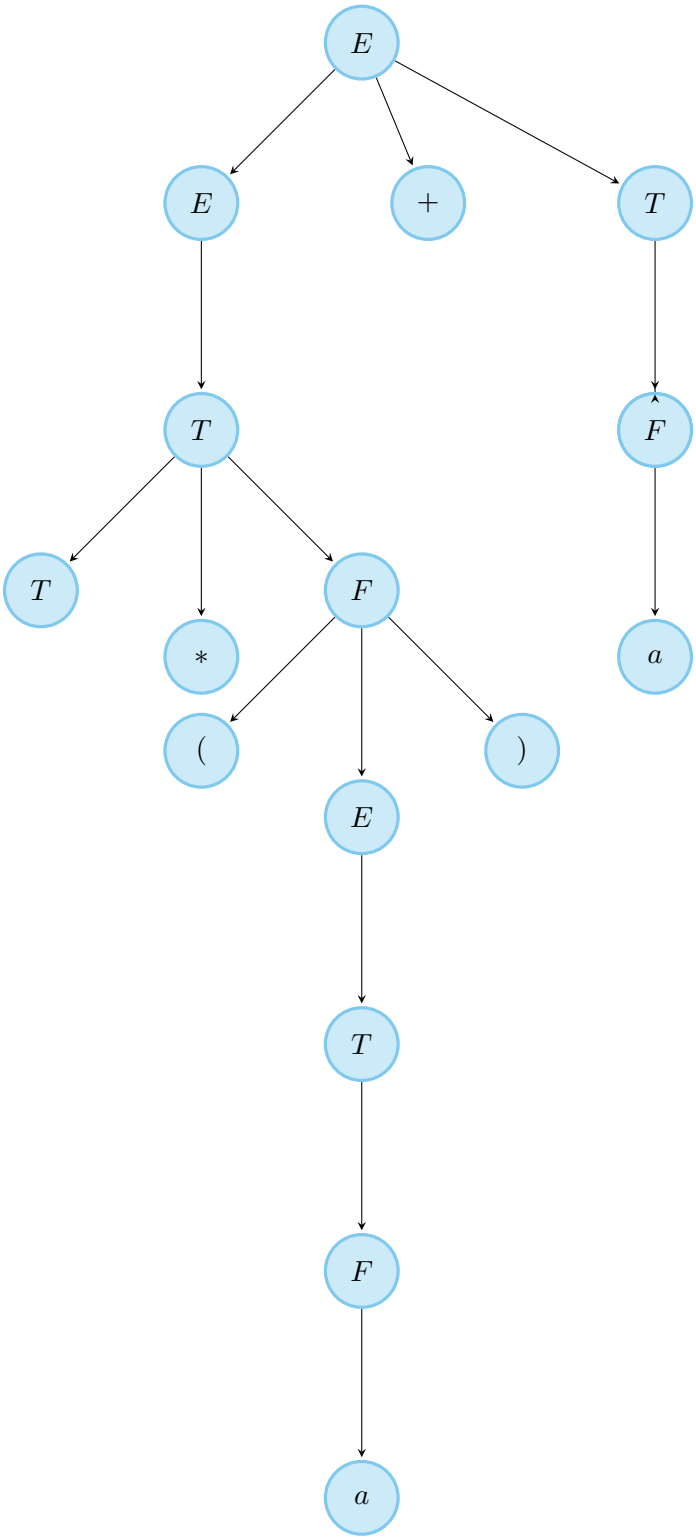
$$\begin{aligned}
E &\implies \mathbf{E} + T \ (1) \\
&\implies T + \mathbf{T} \ (2) \\
&\implies \mathbf{T} + F \ (4) \\
&\implies \mathbf{T} * F + F \ (3) \\
&\implies F * \mathbf{F} + F \ (4) \\
&\implies F * (E) + \mathbf{F} \ (5) \\
&\implies F * (\mathbf{E}) + a \ (6) \\
&\implies \mathbf{F} * (T) + a \ (2) \\
&\implies a * (\mathbf{T}) + a \ (6) \\
&\implies a * (\mathbf{F}) + a \ (4) \\
&\implies a * (a) + a \ (6)
\end{aligned}$$

Hay otras derivaciones de la palabra $a * (a) + a$ en nuestra gramática. Podríamos poner la regla de reemplazar siempre el no-terminal mas a la izquierda (derivación extrema izquierda o canónica) o el más a la derecha (derivación de extrema derecha).

7.6.2. Ejemplo derivación extrema izquierda

$$
\begin{aligned}
E &\implies \mathbf{E} + T \ (1) \\
&\implies \mathbf{T} + T \ (2) \\
&\implies \mathbf{T} * F + T \ (3) \\
&\implies \mathbf{F} * F + T \ (4) \\
&\implies a * \mathbf{F} + T \ (6) \\
&\implies a * (\mathbf{E}) + T \ (5) \\
&\implies a * (\mathbf{T}) + T \ (2) \\
&\implies a * (\mathbf{F}) + T \ (4) \\
&\implies a * (a) + \mathbf{T} \ (6) \\
&\implies a * (a) + \mathbf{F} \ (4) \\
&\implies a * (a) + a
\end{aligned}
$$

7.6.3. Arbol de derivación



Es claro que un árbol de derivación corresponde a múltiples derivaciones, pero un árbol de derivación corresponde a una **única derivación de extrema izquierda (o derecha)**.
Ambiguedad:
Consideremos la gramática:

$$E \to E + E \ (1)$$

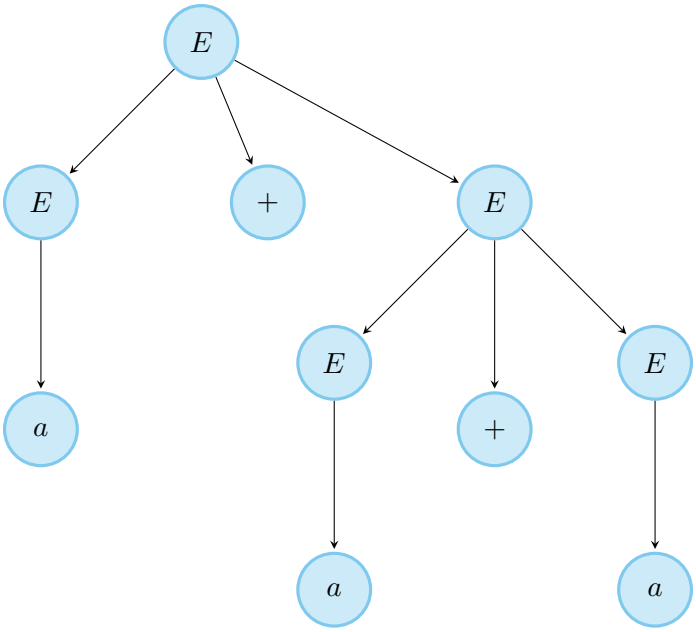
$E \to E * E$ (2)

$E \to (E)$ (3)

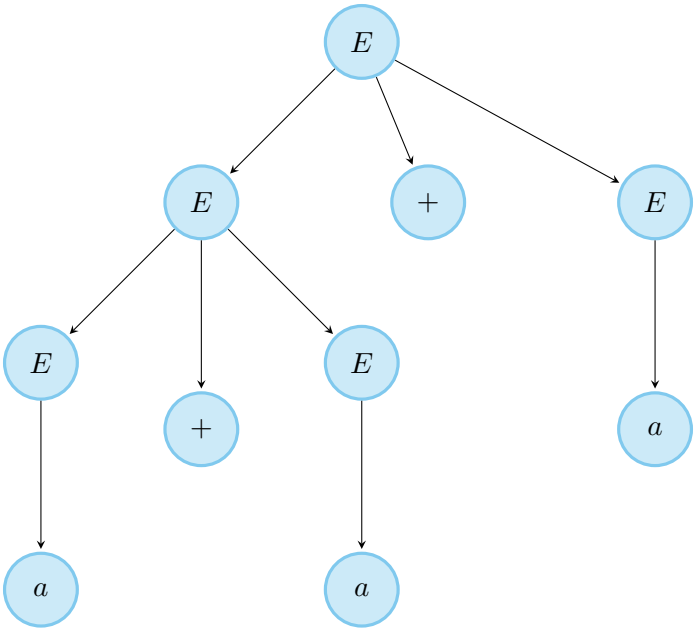
$E \to a$ (4)

Genera el mismo lenguaje que el ejemplo que vimos anteriormente. Pero esta tiene un problema, la palabra $a + a * a$ tiene dos árboles de derivación.

Opción 1



Opción 2



Una gramática para la cual una palabra en el lenguaje tiene mas de un árbol de derivación se dice **ambigua**.

Lamentablemente determinar si una gramática dada es ambigua, no es decidible.

Si un lenguaje de contexto libre tiene gramáticas ambiguas, se dice **inherentemente ambiguo**.

7.7. Forma normal de Chomsky (CNF)

La CFG G se dice en CNF si todas sus producciones tienen las formas:

$$A \to a$$

Con $A \in N$, $a \in \Sigma$

$$A \to BC$$

Con $A, B, C \in N$

Teorema: Sea G una CFG, podemos construir una CFG G' en CNF tal que $L(G') = L(G)$.
Demostración: Varios pasos

- Eliminar producciones unitarios, $A \to B$
- Para cada no-terminal $A \in N$ identificar los no terminales B, tales que $A \implies {}^*B$.
  En el ejemplo, para E:

  $$\{E\}, \{E, T\}, \{E, T, F\}$$

- Eliminar producciones unitarios, remplazar por resultados finales.

  $$E \to T * F$$

  $$E \to (E)$$

  $$E \to a$$

  $$T \to (E)$$

  $$T \to a$$

  En el ejemplo queda:

  $$E \to E + T$$

  $$E \to T * F$$

  $$E \to (E)$$

  $$E \to a$$

  $$T \to T * F$$

  $$T \to (E)$$

  $$T \to a$$

  $$F \to (E)$$

  $$F \to a$$

- Crear nuevos no-terminales X_a para $a \in \Sigma$, con única producción:

  $$X_a \to a$$

  En la gramática, reemplazar a en lados derechos de largo mayor 1 por X_a.

- Producción de lado derecho de largo mayor a 2, sea:

  $A \to B_1, B_2, \dots B_n$

  Agregamos nuevos no-terminales $X_1, X_2, \dots, X_{n-1}$ y reemplazamos por:

  $A \to B_1 X_1$

  $X_1 \to B_2 X_2$

  $\vdots$

  $X_{n-2} \to B_{n-1} B_n$

7.8. Simplificar CFG

En la práctica se aplicaría antes ir a CNF y otros, mas bien se usa para verificar que no queden "cabos sueltos".

7.8.1. Eliminar Símbolos inútiles

(no participa en la generación de palabras del lenguaje)
Un símbolo es útil si es generante y alcanzable:

1. Decimos que $x \in V$ es generate si $X \implies {}^*\omega$ con $\omega \in \sigma^*$
2. Decimos que X es alcanzable si $S \implies {}^*\alpha X \beta$ para algunos α y β.

Las transformaciones siguientes deben aplicarse en el orden indicado

1. Determinar símbolos generates: Estrategia típica de partir con una subestimación e ir agregando faltantes. Claramente, los símbolos en Σ son generantes. Luego, si $V^{(i)}$ es nuestro conjunto de símbolos generantes ($V^{(0)} \in \Sigma$), $V^{(i+1)}$ es $V^{(i)}$ más todos los no-terminales con producciones cuyos lados derechos están formados sólo por símbolos en $V^{(i)}$. Cuando $V^{(k+1)} = V^{(k)}$, paramos. $V^{(k)}$ es V', el vocabulario generante, "podamos " la gramática:

   $G' = (\Sigma, N', P', S)$

   con $N' = N \cap V'$, P' solo producciones en que aparecen símbolos de V'.
   Ejemplo (La regalona)

   $E \to E + T | T$

   $T \to T * F | F$

   $F \to (E) | a$

   Entonces:

   $\Sigma = \{a, (,), *, +\}$

   $N = \{E, T, F\}$

   $V^{(0)} = \Sigma = \{a, (,), *, +\}$

   $V^{(1)} = \{a, (,), *, +, F\}$

   $V^{(2)} = \{a, (,), *, +, F, T\}$

   $V^{(2)} = \{a, (,), *, +, F, T, E\}$

2. Eliminar no alcanzables (misma idea anterior) S es alcanzable: $V^{(0)} = \{S\}$, $V^{(i+1)}$ es $V(i)$ con aquellos símbolos que aparecen en lados derechos de producciones para símbolos en $V^{(i)}$.
   Si $V^{(k+1)} = V^{(k)}$, este es nuestro V'', entonces:

   $G'' = (N'', \Sigma'', P'', S)$

   con $N'' = N' \cap V''$, $\Sigma'' = \Sigma \cap V''$, $P'' : P_0$ dado

7.8.2. Eliminar producciones epsilon

Si hay una producción $A \to \epsilon$, la gramática no es CFG, es tipo 0 (ni siquiera es sensible al contexto). Sin embargo "CFG" con estas producciones generan lenguajes de contexto libre (salvo diferencias triviales).
Podemos eliminar producciones $A \to \epsilon$ mediante la idea anterior:
Un no-terminal X se llama nullable si:

$$X \implies {}^*\epsilon$$

Igual que antes:

- Si $X \to \epsilon$ es una producción, $X \in N^\epsilon$
- Si $X \to AB \dots Z$ es una producción y $A, B, \dots Z \in N^\epsilon$, entonces $X \in N^\epsilon$

Ejemplo

$$E \to TT^+$$

$$T^+ \to +TT^+ | \epsilon$$

$$T \to TF^*$$

$$F^* \to *FF^* | \epsilon$$

$$F \to (E) | a$$

En nuestro caso pertenecen a N^ϵ por derecho propio T^+, F^*. No hay producciones con lado derecho formado sólo por T^+, F^* para el proceso.
Enseguida para cada producción:

$$x \to A_1 A_2 \dots A_n$$

y cada no-terminal nullable crea producciones adicionales eliminando ese no-terminal.
Supongamos nullable $x_1, \dots$ no nullable $A_1, \dots, a_1, \dots$ la producción:

$$A \to A_1 \underline{x_1 x_2} a_1 \underline{x_3} A_2 x_1$$

da:

$$A \to A_1 x_2 a_1 x_3 A_2 x_1$$

$$A \to A_1 x_1 a_1 x_3 A_2 x_1$$

$$\vdots$$

$$A \to A_1 a_1 x_3 A_2 x_1$$

$$\vdots$$

$$A \to A_1 a_1 A_2$$

Finalmente eliminar producciones $X \to \epsilon$, salvo $S \to \epsilon$ (si aparece $S \in N^\epsilon$)

7.9. Lema de bombeo, CFL

Si L es un CFL, hay una constante N tal que si $\sigma \in L, |\sigma| \geq N$ podemos escribir:

$$\sigma = uvxyz$$

Con $|vxy| \leq N$, $vy \neq \epsilon$, tales que para todo $k \geq 0$:

$$uv^k xy^k z \in L$$

7.9.1. Demostración

Sea $L = L(G)$ donde G está en CNF. Sea $\sigma \in L$, como G está en CNF, el árbol de derivación de σ es (más o menos) un árbol binario con $|\sigma|$ hojas. Por lo tanto, la altura del árbol es mayor o igual a $\log_2 |\sigma|$.
Sea n el número de no-terminales de G. Si $|\sigma > 2^n$, sabemos que la altura del árbol de derivación es menor a n y en un camino en el árbol de derivación hay un no-terminal que se repite.
En derivaciones:

$$\begin{aligned} S &\implies {}^* uA^{(1)}z \\ S &\implies {}^+ uvA^{(2)}yz \\ S &\implies {}^+ uvxyz \end{aligned}$$

Por lo tanto son válidas en G:

$$\begin{aligned} S &\implies {}^* uAz \\ S &\implies {}^+ uxz \\ S &\implies {}^* uAz \\ S &\implies {}^* uAz \\ S &\implies {}^+ uvAyz \\ S &\implies {}^+ uv^2Ay^2z \\ &\vdots \\ S &\implies {}^+ uv^kAy^kz \\ S &\implies {}^+ uv^kxy^kz \end{aligned}$$

7.9.2. Ejemplo

El lenguaje $L = \{a^n b^n c^n : n \geq 1\}$ no es de contexto libre. **Demostración**
Por contradicción, si L es CFL, cumple el lema de bombeo, sea N la constante del lema, consideraremos:

$$\sigma = a^N b^N c^N$$

$|\sigma| = 3N \geq N$, por el lema de bombeo podemos escribir:

$$\sigma = uvxyz$$

con $vy \neq \epsilon$ tal que $uv^k xy^k z \in L$ para todo k y $|vxy| \leq N$.
Deben ser v, y formados por un único símbolo. Repitiendo v, y aumenta el número de a a lo más **dos símbolos**, pero debe aumentar en tres.
Por lo tanto existe una contradicción

8. Autómata de Stack (PDA)

Un PDA es un NFA que adicional,ente manipula un stack, considerando el símbolo del tope del stack en transiciones. Formalmente un PDA M consta de:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

Donde: Q : Conjunto finito de estados
Σ :Alfabeto de entrada
Γ : Alfabeto de stack
δ : Función de transición
$\delta : Q \times (\Sigma \cup \{\epsilon\} \times \Gamma \to 2^{Q \times \Gamma^*})$ Subconjuntos finitos
$q_0 \in Q$ Estado Inicial
$Z_0 \in \Gamma$ Stack Inicial
$F \subseteq Q$ Estados Finales
Lo siguiente:

$$(P, \alpha) \in \delta(q, X, Z)$$

Significa que si M está en el estado q, consumiendo X de la entrada, si el tope del stack es Z, pasa al estado P y reemplaza Z por α.

8.1. Descripción Instánea

Sea $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA, una descripción instantánea de $M(ID)(q, \sigma, \gamma)$, consta de un estado $q \in Q$, entrada por leer $\sigma \in \Sigma^*$ y contenido del stack $\gamma \in \Gamma^*$. Definimos la relación de transición de M sobre ID_s.

8.2. Lenguaje aceptado

El lenguaje aceptado por $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es:

$$L(M) = \{\sigma \in \Sigma^* : (q_0, \sigma, z_0) \vdash_M^* (q_f, \epsilon, \gamma), q_f \in F\}$$

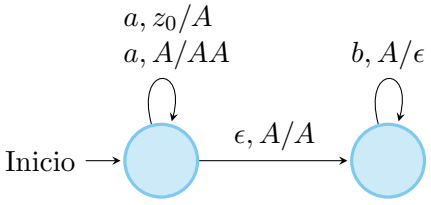
El lenguaje aceptado por M por stack vacío es:

$$N(M) = \{\sigma \in \Sigma^* : (q_0, \sigma, z_0) \vdash_M^* (q_f, \epsilon, \epsilon)\}$$

8.3. Notación Gráfica

$$(x, z/\gamma) \to (entrada, stack/n^o stack)$$

Ejemplo



8.4. Teorema

Si $L = L(M)$ para un PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, podemos construir un PDA M' tal que $L = N(M')$.
La idea es que si M llega a un estado final, dar a M' la opción de *comerse* el resto del stack. Debemos cuidarnos de aceptar accidentalmente, terminando con el stack vacío en un estado no final.

8.4.1. Solución 1

Agregar un nuevo estado q_m con transiciones:

$\delta'(q_f, \epsilon, A) \ni (q_m, \epsilon)$Ir a qm desde cualquier estado final

$q_\epsilon \in F, A \in \Gamma'$

$\delta'(q_m, \epsilon, A) = \{(q_m, \epsilon)\}$Considere el stack completo

8.4.2. Solución 2

Evitar stack vacío accidental, agregar nuevo estado q_0', nuevo stack inicial z_0' y transición:

$\delta'(q_0', \epsilon, z_0') = \{(q_0, z_0', z_0)\}$

En la primera movida M' deja z_0' z_0 en el stack y q_0 en el estado. Luego sigue las movidas de M que nunca pueden borrar z_0' (no lo mencionan), solo se puede eliminar z_0' mediante $\delta'(q_m, \epsilon, z_0') = \{(q_m, \epsilon)\}$ parte de M. $\square$.

Idea: Queremos que si M tiene stack vacío, M' vaya a un estado final, nuevamente agregamos nuevo comienzo q_0', z_0' con:

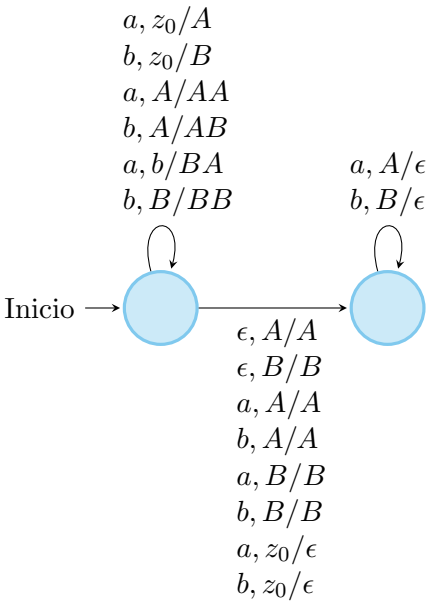
$\delta'(q, \epsilon, z_0') = \{(q_f, \epsilon)\}\text{con}F = \{q_f\}$

8.5. PDA Determinista (DPDA)

El PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es determinista si:

- $|\delta(q, X, A)| \leq 1 \forall q \in Q, X \in \Sigma \cup \{\epsilon\}, A \in \Gamma$
- Si $\delta, q, \epsilon, A \neq \phi$, entonces $\delta(q, a, A) = \phi \forall a \in \Sigma$

Hay lenguajes que un PDA no determinista reconoce, pero que ningún $DPDA$ reconoce.
Consideremos el lenguaje de palíndromos $\{\omega\omega^k : \omega \in \Sigma^*\} \cup \{\omega a \omega^k : \omega \in \Sigma^* \wedge a \in \Sigma\}$, si $\Sigma = \{a, b\}$ por stack vacío:



No hay nada que marque el punto central de $\omega\omega^k$, un $DPDA$ no puede decidir cuando cambiar de acumular (leyendo ω) a verificar (leyendo ω). $\square$

8.5.1. Equivalencia entre CFL y PDA

Sea $G = (N, \Sigma, P, S)$ una CFG, entonces podemos construir un PDA M tal que $L(G) = N(M)$. Construiremos el PDA $M = \{\{q_0\}, \Sigma, N \cup \Sigma, \delta, q_0, S, \phi\}$ con las siguientes transiciones (suponiendo que el stack crece hacia la izquierda):

- $\delta(q_0, a, a) = \{q_o, \epsilon\}$ para todo $a \in \Sigma$ (Consume terminales generados)
- Si $A \to \alpha$ es una producción, entonces $(q_0, \alpha) \in \delta(q_0, \epsilon, A)$ (Expande no-terminales en el stack)

Resulta que $L(G) = N(M)$. $\square$.
(Notar que M tiene un único estado).
Ejemplo

1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to a$

Entrada	Stack	Producción
$a+(a+a)*a$	E	1
$a+(a+a)*a$	$E+T$	2
$a+(a+a)*a$	$T+T$	4
$a+(a+a)*a$	$F+T$	6
$a+(a+a)*a$	$a+T$	
$+(a+a)*a$	$+T$	
$(a+a)*a$	T	3
$a+(a+a)*a$	$T*F$	4
$a+(a+a)*a$	$F*F$	5
$a+(a+a)*a$	$(E)*F$	
$a+a)*a$	$E)*F$	1
$a+a)*a$	$E+T)*F$	2
$a+a)*a$	$T+T)*F$	4
$a+a)*a$	$F+T)*F$	6
$a+a)*a$	$a+T)*F$	
$+a)*a$	$+T)*F$	
$a)*a$	$T)*F$	4
$a)*a$	$F)*F$	6
$a)*a$	$a)*F$	
$)*a$	$)*F$	
$*a$	$*F$	
a	F	6
a	a	
ϵ	ϵ	

Hay una manera de extender esto a un $DPDA$ para ciertas gramáticas: $LL(k)$ (Left to right, left most derivation, k symbols lookahead). De importancia práctica: Gramáticas LL(1) para las que es posible escribir un programa *a mano*.
Estrategia *top-down* ($\rightarrow$ descenso recursivo, backtracking) cada no-terminal es una función que lee de la entrada los terminales y llama a las funciones para reconocer no terminales en su expansión.

8.6. PDA extendido (XPDA)

En vez de considerar un símbolo en el stack, se pueden considerar varios, tomando que el stack crece $\rightarrow$.
Un $XPDA$ $M = (Q, \Sigma, \Gamma, \delta, q_o, Z_0, F)$ es como un PDA pero:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^+ \rightarrow 2^{Q \times \Gamma^*}$$

con la restricción que δ sólo está definida para un conjunto finito de $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^+$.
Teorema PDA y XPDA reconocen los mismos lenguajes.
Demostración:
PDA es un caso particular de XPDA, podemos simular una movida del XPDA con varias movidas de un PDA. Por ejemplo:

$$(p, \alpha) \in \delta(q, x, A_1, A_2, \ldots, A_n)$$

Se simula por:

$$\delta'(q, x, A_n) = \{(qA_2 \dots A_{n-1}, \epsilon)\}$$
$$\delta'(qA_1 \dots A_{n-1}, \epsilon, A_{n-1}) = \{(qA_2 \dots A_{n-2}, \epsilon)\}$$
$$\vdots$$
$$\delta'(qA_1, \epsilon, A_1) = \{(p, \alpha)\}$$

Podemos usar XPDA para construir un árbol de derivación *desde las hojas* vía acumular símbolos en el stack (shift) y aplicar producciones en reversa (reducir).

8.6.1. Ejemplo

Ejemplo con gramática regalona;

1. $E \to E + T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to a$

Entrada	Stack	Op
$a + (a + a) * a$	z_0	shift a
$+(a + a) * a$	a	Reduce 6
$+(a + a) * a$	F	Reduce 4
$+(a + a) * a$	T	Reduce 2
$+(a + a) * a$	E	Shift +
$(a + a) * a$	$E+$	Shift (
$a + a) * a$	$E + ($	Shift a
$+a) * a$	$E + (a$	Reduce 6
$+a) * a$	$E + (T$	Reduce 2
$+a) * a$	$E + (E$	Shift +
$a) * a$	$E + (E+$	Shift a
$) * a$	$E + (E + a$	Reduce 6
$) * a$	$E + (E + F$	Reduce 4
$) * a$	$E + (E + T$	Reduce 1
$) * a$	$E + (E$	Shift)
$*a$	$E + (E)$	Reduce 5
$*a$	$E + F$	Reduce 4
$*a$	$E + T$	Shift $*$
a	$E + T*$	Shift a
ϵ	$E + T * a$	Reduce 6
ϵ	$E + T * F$	Reduce 3
ϵ	$E + T$	Reduce 1
ϵ	$E * a$	Acepta
ϵ	z_0E	

El XPDA acepta por stack vacío, en el proceso traza una derivación de extrema derecha en reversa.

8.6.2. Dos ideas

- **Top-Down** Expandir no-terminales, consumir terminales resultantes. Construye una derivación de extrema izquierda.
- **Bottom Up** Acumular símbolos en el stack (shift), reducir lados derechos de producciones a los lados izquierdos. Construye una derivación de extrema derecha en reversa.

Llevadas a autómatas deterministas: **Top Down** Gramáticas LL(k), viendo los k símbolos siguientes, podemos determinar la alternativa del no-terminal que se expande. El caso típico es con $k = 1$, útil para hacer a mano (descenso recursivo)
Bottom Up Gramáticas LR(k) viendo el contenido del stack (lado derecho de una producción o parte de esto) y k símbolos, decide determinísticamente si seguir acumulando (shift) o reducir. Caso más útil $k = 1$.

8.7. CFG a partir de un PDA

Sea $M = (Q, \Sigma, \Gamma, \delta, q_o, Z_0, F)$ un PDA, entonces podemos construir un CFG G tal que $N(M) = L(G)$.
Demostración
Cuando M procesa una palabra llegará a alguna situación en la cual (el stack crece hacia la izquierda) el stack contiene $A\alpha$ con $\alpha \in \Gamma^*$, $A \in \Gamma$ y está en el estado q. En las movidas siguientes habrá una situación en que por primera vez en el stack queda sólo α (consumió A del stack), quedando en el estado p. Nótese que como en el stack nunca hay nada de α en el tope, α no influye en este proceso, M consume algún string σ de la entrada.
La idea es definir no-terminales $[q, A, p]$ $p, q \in Q, A \in \Gamma$ tales que si M, partiendo del estado q, consume A del stack y queda en el estado p, consumiendo σ de la entrada:

$$[q, A, p] \implies {}^*\sigma$$

Vale decir, $N(M)$ es el conjunto de strings tales que para algún $q' \in Q$ tenemos:

$$[q_0, Z_0, q'] \implies {}^*\sigma$$

Añadimos un no-terminal S como símbolo de partida:

$$S \to [q_o, Z_0, q'] \forall q' \in Q$$

Consideremos:

$$(p, \gamma) \in \delta(q, x, A)$$

con $\gamma = A_1 A_2 \dots A_n$
Si tomamos este paso, quedamos en el estado p con el compromiso de consumir $A_1 A_2 \dots A_n$ del stack. Al final de este proceso estaremos en algún estado, llamemosle q, Esto sugiere producciones:

$$[q, A, q_n] \to x[p, A_1, q_1][q_1, A_2, q_2] \dots x[q_{n-1}, A_n, q_n]$$

Esto no es una producción, son muchas, es decir, una para cada combinación $(q_1, \dots, q_n) \in Q^n$, por ejemplo si $\gamma = \epsilon$, o sea $n = 0$, entonces:

$$[q, A, p] \to x$$

La CFG con estos no-terminales Σ de M, estas producciones y símbolo de partida S generan $N(M)$.

8.7.1. Algunas Conclusiones adicionales

- Para una CFG construimos un PDA con un único estado que acepta por stack vacío.
- Para un PDA construimos una CFG.
- Podemos aceptar cualquier CFL con un PDA con 1 estado.

8.8. Propiedades de Clausura de CFL

1. Cerradas respecto a la substitución.
2. Cerradas respecto de homomorfismo.
3. Cerrados respecto de homomorfismos inversos.
4. Cerrados respecto de intersección con lenguajes regulares.

Demostración:
Sea $L_1 = L(M)$, $L_2 = L(M2)$ con los PDA:

$$M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_1, F_1)$$
$$M_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_2, Z_2, F_2)$$

Definimos el PDA:

$$M = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), Z_1, F_1 \times F_2)$$

Donde:

$$\delta((q', q''), x, A) \ni ((p', p''), \gamma)$$

Si $\delta_1(q', x, A) \ni (p', \gamma)$ y $\delta_2(q'', x) = p''$ si $x \neq \epsilon$, $p'' = q''$ si $x = \epsilon$. (Simula M_1 con el primer componente del estado y stack, simula M_2 con el segundo componente del estado). $\square$.
Nótese que no son cerrados respecto a intersección:

$$L_1 = a^m b^m c^n : m, n \geq 1$$
$$L_2 = a^m b^n c^n : m, n \geq 1$$

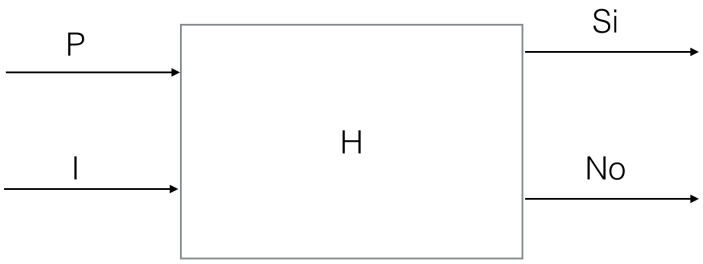
Pero $L_1 \cap L_2 = a^n b^n c_n^n \geq 1$ no es CFL.
Como no son cerrados respecto intersección, tampoco son cerrados respecto a complemento. (cerrado respecto a unión y complemento implica que es cerrado respecto a intersección, son cerrados respecto unión, pero no intersección).

9. Computación

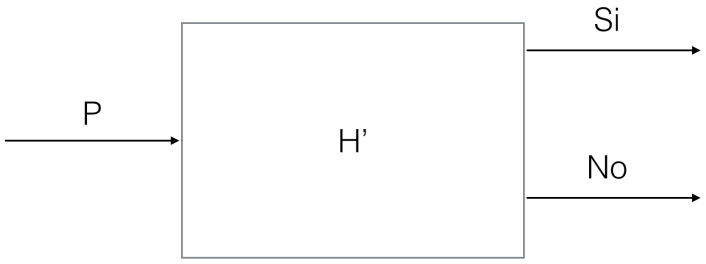
Computar: Tomar una entrada (string) y dar una respuesta (string), un algoritmo sobre strings es natural considerado un autómata.

9.1. Modelo de Computación

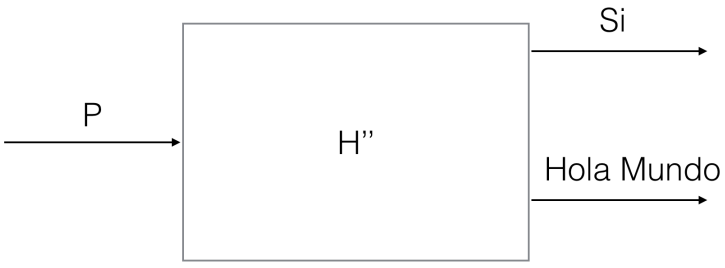
Teorema
No hay detector de *Hola Mundo*
Detector de Hola Mundo Un programa recibe de entrada el programa P y datos de entrada I, y en un plazo finito, sin equivocarse nunca, dice *Si* si P con entrada I escribe *Hola Mundo* como primera cosa, *No* en caso contrario. **Demostración**
Por contradicción:
Supongamos que existe el programa H



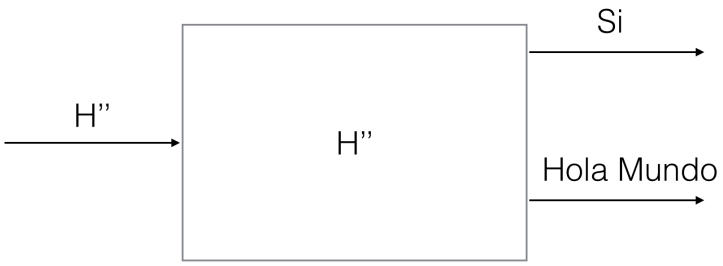
Tomamos el supuesto programa H, lo modificamos de forma que analice P con datos de entrada el código de P.



Ahora modificamos H' de forma que si H' escribe *No*, escriba *Hola Mundo*



¿Qué ocurre si alimentamos H'' con código de H''?



No puede responder *Si* ni *Hola mundo*, lo cual es una contradicción. Un problema P es un lenguaje, buscamos responder la pregunta ¿$\sigma \in P$? (donde σ es una instancia de P)

9.2. Reducción

Sea P y Q problemas, una reducción del problema P al problema Q es un algoritmo (programa que se detiene siempre en tiempo finito) que *traduce* toda instancia α de P a una instancia β de Q con la misma respuesta. Se anota $P \to Q$. Esto significa que podemos responder a ¿$\alpha \in P$? vía traducir α en β y luego resolver ¿$\beta \in Q$?.
La idea realmente es usar esto para demostraciones por contradicción:
Para demostrar que Q no puede resolverse, demostramos $P \to Q$, y P no puede resolverse con $P \to Q$ estamos demostrando que Q es a lo menos tan difícil de resolver como P. Por esto, muchos anotan $P \leq Q$ en vez de $P \to Q$.

9.3. Problema no decidible

Un problema que ningún algoritmo puede resolver se llama no decidible.

9.3.1. Ejemplos

Sea P un programa en C¿Alguna vez se le asigna un valor a la variable x ?
La idea es reducir la detección de *Hola, mundo!* a este problema.
Sea A un programa C cualquiera. Lo modificaremos de manera que asigne a x si y solo si A escribe *Hola, mundo!*. Si es así, ejecuta $x = 1$ (asigna valor a la variable x)

9.4. Máquinas de Turing

1936, Alan Turing *Modelo de toda computación poisble*

$M = (Q, \Sigma, \delta, q_0, B, F)$

- Q: Conjunto finito de estados
- Σ: Alfabeto de entrada
- Γ: Alfabeto de cinta, $\Sigma \subset \Gamma$
- δ: Función de transición

  $$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times I,D}$$

- q_0: Estado inicial, $q_0 \in Q$
- B: Símbolo blanco, $B \in \Gamma$, $B \notin \Sigma$
- F: Estados Finales $F \subseteq Q \to \delta(x, q_f) = \phi$ si $q_f \in F$ (si llega a un estado final se detiene)

9.4.1. Descripción Instantanea (ID)

Suponemos $Q \wedge \Gamma = \phi$

$\alpha q \beta$

con $\alpha, \beta \in \Gamma^*, q \in Q$.
La idea es que M *mira* el primer símbolo de β o B si $B = \epsilon$

9.4.2. Relación de Transición

$\alpha q \beta \vdash_M \alpha' q' \beta'$

9.4.3. Lenguaje aceptado por la Maquina de Turing

$L(M) = \sigma : q_o \sigma \vdash_M^* \alpha q_f \beta, q_f \in F$

También como calcular una función:
$f_u(\sigma)$ lo que queda en la cinta cuando M se detiene al partir con σ.

9.4.4. Descripción de la Máquina de Turing

Restringimos:

$$\begin{aligned} Q &: 1 \dots N \\ q_0 &= 1 \\ F &= \{2\} \\ \Sigma &: 2 \dots M \\ \Gamma &= M+1 \dots k \\ B &= 1 \end{aligned}$$

Se puede escribir δ como una lista:

$q\#a\#p\#b\#1$ para I, 2 para D

Con las convenciones de antes puedo describir M como la función δ.
Si escribo números naturales como $1, 11, \dots$ y uso 0 como separación (#), una TM queda descrita por una palabra en binario. Esta la puedo interpretar como un número binario, *el mismo de la TM*, bajo el supuesto que hay números en binario que no cumplen el formato correspondiente a la TM que siempre se mueve a la derecha, puedo hablar de la TM número n.

9.4.5. Lenguaje Diagonal

Imaginemos una matriz infinita:

$$\begin{bmatrix} TM & 1 & 2 & 3 & \dots \\ 1 & & & & \\ 2 & & & & \\ 3 & & & & \\ \vdots & & & & \\ M < M > & \dots & \dots & \{0,1\} & \end{bmatrix}$$

Donde (i, j) es 0 si M_j no acepta σ_j y 1 si M_j acepta σ_j . Entonces el Lenguaje diagonal es:

$$L_d = \{i : M_i no acepta \sigma_j\}$$

Por lo tanto L_d no es aceptado por ninguna TM.
Demostración: Por contradicción, supongamos que:

$$L(M) = L_d \text{con M una TM}$$

La TM tiene codigo k con lo que $\sigma_k \in L_d$, pero por definición de L_d $\sigma_k \notin L_d$, es decir, se genera una contradicción.

9.4.6. Maquinas de Turing enchuladas

1. Varias pistas en vez de estar divididas en cuadrados.
   La forma simple, si hay p pistas:

   $$\Sigma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_p$$

   El truco típico es usar una pista adicional con marcas, para por ejemplo marcar donde está.

2. Cinta infinita en ambas direcciones:
   Usar dis pistas, arriba $\rightarrow$ y abajo$\leftarrow$, marcar inicio, registrar en el estado, si está arriba o abajo.

3. Varios cabezales:
   Una pista para la cinta, pistas adicionales para cada cabezal que marcan su posición.
   Una movida de la TM multicabezal se simula con un ciclo, comenzando a la izquierda de todo los cabezales.

   - Recorrer hacia la derecha, recogiendo en el estado los símbolos vistos por cada cabezal.
   - Devolverse a la posición del primer cabezal
   - Avanzar, cambiando el símbolo re escrito por cada cabezal, mover(si corresponde) la marca que da la posición del cabezal.
   - Cambiar el estado simulado
   - Volver a la posición del primer cabezal.

4. Multiples cintas:
   La TM con k cintas se simula con $2k$ pistas, cada cinta se representa por una pista con un contenido y otra con una marca indicando la posición del cabezal.

Un programa (Máquina de Turing) puede describirse mediante un string sobr eun alfabeto adecuado.

- El conjunto de programas es numerable
- El conjunto de lenguajes sobre Σ no es numerable

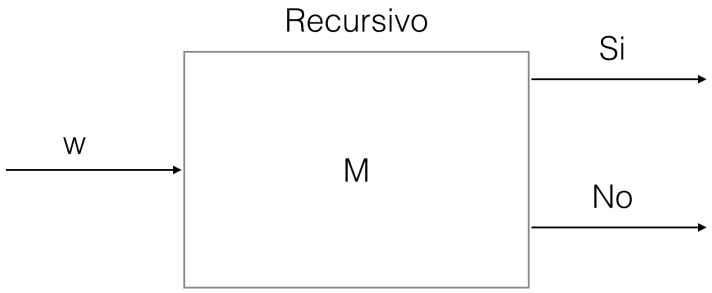
Entonces hay lenguajes que no son reconocidos por ninguna Máquina de Turing.
Representar Máquina de Turing mediante números, representar $\sigma \in \Sigma^*$ mediante números. Existe una biyección entre entradas de Máquina de Turing y números naturales ($\mathbb{N}$). Dada M, anotamos $< M >$ para su número, dado σ, anotamos $< \sigma >$, la combinación M, σ por $< M, \sigma >$.

9.4.7. Ejemplo Máquina de Turing

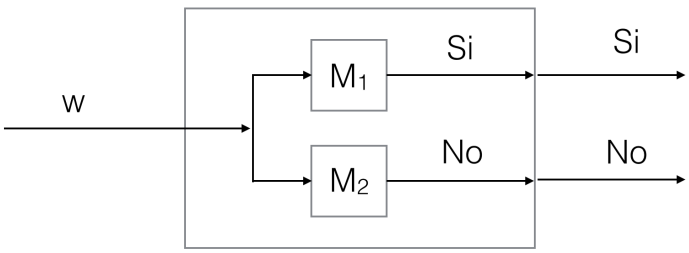
Con $a^n b^n : n \geq 0$
aaabbbb
(falta dibujo)
Con la enumeración:
(falta tabla) Podemas construir el lenguaje Ld:

$$Ld = < M >:< M > noacepta < M >$$

No hay máquina de Turing que acepte Ld.
Un lenguaje es recursivo si hay una TM que siempre se detiene que acepta/rechaza.
Un lenguaje es recursivamente enumerable (R.E) si hay una TM que lo acepta (Si no acepta, puede que no se detenga)



Teorema Si L es $R.E$ y $\bar{L}$ es R.E entonces L (y $\bar{L}$) es recursivo.
Demostración:
Supongamos $L = L(M_1)$ y $\bar{L} = L(M_2)$, entonces:



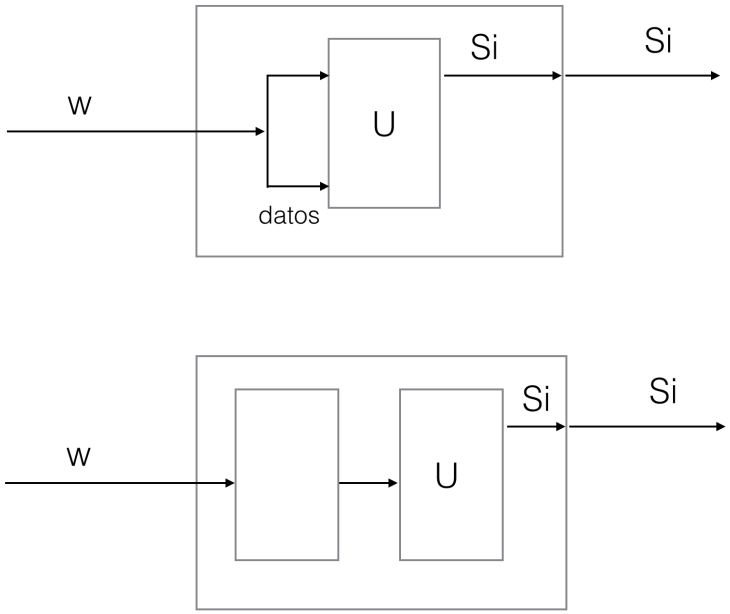
Corolario:
Si L es $R.E$ pero no recursivo $\bar{L}$ no es $R.E$.
Posibilidades:

L	$\bar{L}$	¿Posible?
Rec	Rec	Si
Rec	R.E	Si, $\bar{L}$ es recursivo
Rec	no RE	No
RE	RE	Si L y $\bar{L}$ recursivos
no RE	no RE	Si L no es recursivo

L_d no es $R.E$, entonces:

$\bar{L_d} = < M >$: M acepta ¡M¿

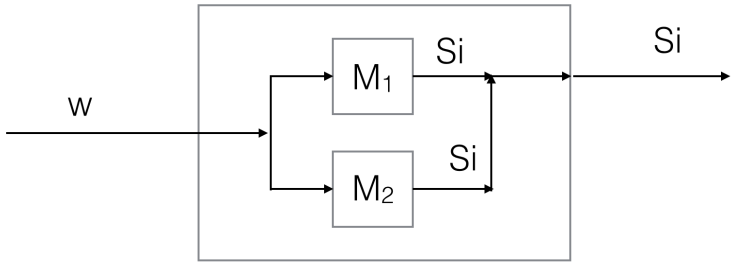
Tenemos la TMU



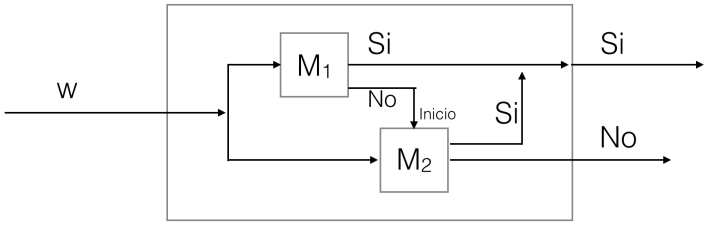
L_M es RE, no recursivo.
Un problema es decidible si es recursivo.
¿Por qué Re?
Sea L RE $(L = L(M))$ podemos construir una TM que haga lo siguiente:

- Genera $j + k$ en orden creciente
- Simula M sobre ω_j por k pasos
- Si M acepta, entonces copia ω_j a la salida

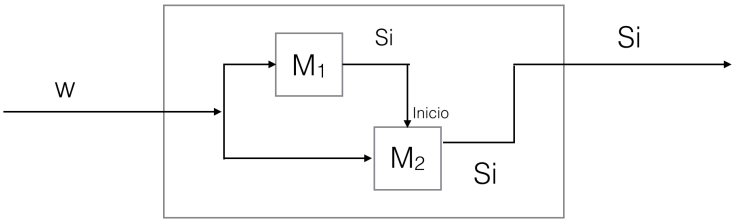
Teorema
El conjunto de lenguajes $R.E$ es cerrado respecto de unión.
Demostración:



Teorema Los lenguajes recursivos son cerrados respecto a unión.
Demostración: $L_1 = L(M_1)$ y $L_2 = L(M_2)$



Respecto a intersección:



9.5. P vs NP

Hay muchos problemas de interés práctico para los cuales el mejor algoritmo conocido toma tiempo exponencial en el tamaño de la entrada.
Vimos que si tenemos modelos de computación alternativos (TM, RAM, ...) con medidas de tiempo (número de movidas, número de instrucciones ejecutadas, ...) podemos simular uno cualquiera de ellos en el otro en un tiempo acotado por un polinomio en el tiempo original.
Resulta que los polinomios son cerrados respecto de composición, si $p(x)$, $q(x)$ son polinomios, entonces también lo es $p(q(x))$.
Diremos que un problema tiene **solución eficiente** si hay una TM determinista que resuelve una instancia de tamaño N (igual largo de la entrada) en $O(N^K)$ pasos, para $k \in \mathbb{N}$.
No determinismo no agrega capacidad de resolver problemas a TM (podemos simular los caminos de la NTM) a lo ancho en forma determinista, si L es aceptado por una NTM es aceptado por una TM). Pero si parece acortar tiempos de cómputo.
Un problema se dice perteneciente a la clase P si hay una TM determinista que siempre se detiene, acepta el lenguaje y que para entrada de largo N toma $O(N^K)$ pasos, con $k \in \mathbb{N}$ (tiempo polinomial).
Se dice que pertenece a NP si hay una TM no determinista que toma $O(N^K)$ pasos, con $k \in \mathbb{N}$ (tiempo no determinista polinomial).

Es claro que $P \subseteq NP$, la pregunta ¿Es $P = NP$? sigue abierta.
Hay muchos problemas de inmenso interés práctico en NP para los que no se conocen algoritmos deterministas que toman tiempo polinomial.

9.5.1. Descripción alternativa de NP

Adivinar la solución (no determinista, tiempo polinomial) y verificar que es solución (determinista, polinomial)

9.5.2. Reducción polinomial

Dada una instancia del problema P_1 se construye en forma determinista en tiempo polinomial una instancia del problema P_2 con la misma respuesta.
(DIBUJO).
Un problema se llama NP duro (NP hard) si todo problema en NP puede reducirse polinomialmente a él. NP completo es un NP duro en NP.
Ejemplo problema NP-completo
Determinar si la NTM M acepta σ en tiempo acotado por $p|\sigma|$ con p un polinomio es NP completo.

9.5.3. SAT

Dada una expresión lógica (de largo N) determina si hay una combinación de valores de las variables que la hace verdadera (satisfacible, $\implies$ SAT)

9.5.4. Forma normal conjuntiva (CNF)

- Átomo: variable (x_i) o su negación ($\bar{x_i}$)
- Cláusula: Disjunción (o, v) de átomos.
- Fórmula: Conjunción $(y, 1)$ de cláusulas

3 CNF: CNF, pero toda cláusula consta de 3 átomos.

9.5.5. Teorema de Cook

3 SAT es NP completo (SAT de fórmulas en 3CNF).
La razón es la siguiente:
SAT se reduce polinomialmente a 3SAT, por lo tanto SAT es NP-duro, como 3SAT es caso particular de SAT entonces 3SAT es NP y por lo tanto NP completo. 2 SAT $\in$ P
Si Q es un problema NP-duro, todos los problemas en NP se pueden reducir polinomialmente a él.
Si Q es NP Completo, es NP-duro y está en NP.

9.5.6. CLIQUE

Dado un grafo $G = (V, E)$ y un entero k, ¿Tiene G un subgrafo K_k?
Reducimos 3SAT a CLIQUE.
Sea Φ una fórmula en 3CNF con k cláusulas, consturimos un grafo G que tiene una k-clique si y sólo si Φ es satisfacible.

9.5.7. Independent SET

Dado el grafo $G = (V, E)$ ¿Hay un conjunto de k vértices que no estén conectados?
Si G tiene un conjunto de tamaño k, $\bar{G}$ tiene una k-clique. CLIQUE se reduce polinomialmente a INDEPENDENT SET.