21 Febrer 05
Elisenda Rovirosa Mora
Q. Primavera 04-05.

TC – Teoría de la computación

Grupo 20
Enrique Romero
C6-210 → Ω-319
eromero@lsi.upc.edu
Horario consultas: Jueves 12-14h

- Temario:

1. Problemas, lenguajes y funciones
2. Máquinas de Turing
3. Máquinas de Turing y algoritmos
4. Computabilidad y decidibilidad
5. Reductibilidad y completitud
6. Problemas indecidibles clásicos
7. Modelos reconocedores y modelos generadores — 1a Part.

---

8. Gramáticas incontextuales — 2a Part.
9. Normalización de gramáticas
10. Autómatas finitos
11. Minimización de autómatas finitos
12. Expresiones regulares y gramáticas regulares
13. Autómatas con pila
14. Problemas clásicos sobre autómatas finitos y gramáticas incontextuales

- Evaluación del curso:

Exámen Parcial → P
Exámen Final → F

* Nota Teoría $T = \max\left(F, \frac{P+F}{2}\right)$
* Nota Ejercicios ⇒ X

$T+X$ si $T \geq 3{,}5$

21 Febrer05

* Introducción Primera Parte

- Modelo de cálculo ⇒ Algo en qué se puede convertir un algoritmo.

Reciben entrada → Algoritmo → Producen salida

↓

Tienen en común que tienen que poderse representar de manera finita.

Pex: ni el num e, ni π poden representar-se de manera finita. → un nombre real en general no es pot representar de manera finita

⇒ Un algoritmo nunca puede recibir una entrada infinita porque sino no se sabe cuando parar de leer.

Lenguajes → conjuntos que se pueden representar de forma finita
≈ cosas potencialmente finitas que se pueden representar de forma finita.

Modelo cálculo < Máquina de Turing, Gram. contextuales, Autómatas

Máquina de Turing < Manera formal para trabajar con los alg.
Qué problemas son resolubles vía alg (Maq. Turing)?

la tendencia es pensar que dado un problema X, existe un alg Y que lo soluciona.
⇒ Però NO. Existen problemas que no tienen solución (límites de la computación – Tema 6.)

* PROBLEMAS NO RESOLUBLES

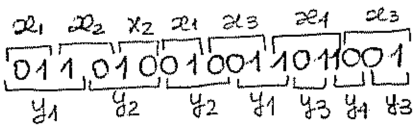
⇒ Ejemplo Problema: Decidir si un programa en C escribe "hello, world".

Planteamiento: Dado un cierto programa como entrada decidir si hace algo

⇒ Este problema no es resoluble vía algoritmo para cualquier algorimo de entrada → siempre dejaremos ∞ casos para cubrir

⇒ Correspondencia de Post: Dadas 2 listas $A = (x_1, \dots x_n)$ y $B = (y_1, \dots, y_n)$ determinar si existe una secuencia de enteros $(i_1, \dots, i_R)$ tal que $x_{i1} \dots x_{iR} = y_{i1} \dots y_{iR}$

	A	B	
x_1	01	011	y_1
x_2	10	010	y_2
x_3	001	01	y_3
x_4	1011	10	y_4



A	B
101	0100
100	10
0110	01
1010	10

No existe salida

El nombre de problemes resolubles mitjançant alg és infinitament menor que el nombre de problemes no resolubles ⇒ la probabilitat tendeix a 0.

• Tema 1 – Problemas, lenguajes y funciones

Estilo de Problemas:

· Accesibilidad de Grafos: Dado un $\boxed{\text{grafo } G=(V,E) \text{ y } 2 \text{ vértices } u,v}$ determinar si v es accesible desde u en G ≅ Existe un camino de u a v ⇒ Problema Decisional (si/no)

entrada ⇒ Representable de forma finita

· Camino: Dado un grafo $G=(V,E)$ y 2 vértices u,v, encontrar un $\boxed{\text{camino}}$ de u a v (si existe) ⇒ Problema Funcional

salida

Uno es caso particular del otro

Definición: un PROBLEMA se caracteriza por 3 elementos

- un conjunto de entradas E
- un conjunto de salidas S
- una propiedad $P(x,y)$ $x \in E, y \in S$

Diremos que una cierta salida y ($y \in S$) es solución del problema con entrada $x \in E$ si $P(x,y)$ es cierto

- Problema Decisional: Dado $x \in E$, ¿Existe $y \in S$ tq. $P(x,y)$ sea cierto?
- Problema Funcional: Dado $x \in E$, encontrar $y \in S$ tq. $P(x,y)$ sea cierto.

* Ejemplos ACCESIBILIDAD

$$E = \left\{ \underset{G=(V,A)}{\overset{\Downarrow}{(G,u,v)}} \;\middle|\; G \text{ es un grafo y } u,v \text{ vértices} \right\}$$

$$\exists K : K \le n : \boxed{\exists \underset{\in V}{u_1, \dots u_K} : u_1 = u \wedge u_K = v \wedge \forall i : 1 \le i < K : (u_i, u_{i+1}) \in A}$$

$P(x,y) \Rightarrow$ $x = G, u, v$
$y = u_1 \dots u_K$

$S = \{ (u_1 \dots u_K) \}$

Lenguaje

Entradas y salidas de longitud finita.
Possibilidad de infinitas entradas y salidas } Lenguajes ⇓ Trabajar con algoritmos

Alfabeto: Es un conjunto finito y no vacío de símbolos Notación: Σ

Palabra: secuencia finita de símbolos yuxtapuestos

Lenguaje: Es cualquier conjunto (finito/infinito) de palabras

· Ejemplos

alfabetos: $\Sigma_1 : \{0,1\}$
$\Sigma_2 : \{a,b,c\}$
$\Sigma_3 : \{a,b,c \dots, z\}$

palabras: sobre $\Sigma_1 \rightarrow \omega_1 : 0110$, $\omega_2 : 10001$, $\omega_3 : 0$
sobre $\Sigma_3 \rightarrow \omega_1 :$ Elis → código fuente.

→ un código fuente

lenguaje: L1: Diccionario de Lengua
L2: {palabras que empiezan por a}

L3: {00, 001, 1001} lenguaje con 3 palabras

21 Febrer05

• Tema 1 : Problemas, Lenguajes y funciones

* Problemas :

entrada ⇒ Representable de forma finito

- Accesibilidad de grafos : Dado un grafo $\boxed{G = (V,E) \text{ y 2 vértices } u, v}$ determinar si v es accesible desde u en G $\cong$ Existe un camino de u a v. ⇒ Problema Decisional (Si/no)
- Camino : Dado un grafo $G=(V,E)$ y 2 vértices u,v encontrar un $\underset{\text{sortida}}{\boxed{\text{camino}}}$ de u a v (si existe) ⇒ Problema Funcional

uno es caso particular del otro. ⇒

un PROBLEMA se caracteriza por 3 elementos :

- un conjunto de entradas E
- un conjunto de salidas S
- una propiedad $P(x,y)$ $x \in E$, $y \in S$

Diremos que una cierta salida y ($y \in S$) es solución del problema con entrada x ($x \in E$) si $P(x,y)$ es cierto.

- Problema Decisional : Dado $x \in E$, ¿Existe $y \in S$ tq. $P(x,y)$ sea cierto?
- Problema Funcional : Dado $x \in E$, encontrar $y \in S$ tq. $P(x,y)$ sea cierto.

▸ Ejemplo Accesibilidad

$$E = \left\{ (G,u,v) \;\middle|\; \begin{array}{l} G \text{ es un grafo } G=(V,A) \\ \text{y } u,v \text{ vértices} \end{array} \right\}$$

$$\exists k : k \leq n : \boxed{\exists u_1, \ldots, u_k \in V : u_1 = u \wedge u_k = v \wedge}$$
$$\boxed{\forall i : 1 \leq i \leq k : (u_i, u_{i+1}) \in A}$$

$$P(x,y) \Rightarrow \begin{array}{l} x = G, u, v \\ y = u_1 \ldots u_k \end{array}$$

$$S = \{ (u_1 \ldots u_k) \}$$

* Lenguaje :

Entradas y salidas de longitud finita
Posibilidad de infinitas entradas y salidas } Lenguajes ↓ Trabajar con algoritmos

Alfabeto : Es un conjunto finito y no vacío de símbolos. Notación Σ
Palabra : Secuencia finita de símbolos yuxtapuestos
Lenguaje : Es cualquier conjunto (finito/infinito) de palabras

▸ Ejemplos :

alfabetos :
$\Sigma_1 : \{0,1\}$
$\Sigma_2 : \{a,b,c\}$
$\Sigma_3 : \{a,b,c,\ldots,z\}$

palabras :
sobre $\Sigma_1 \rightarrow \omega_1 : 0110$, $\omega_2 : 10001$, $\omega_3 = 0$
sobre $\Sigma_3 \rightarrow \omega_1$: Elís
codí font

lenguaje :
L_1 : Diccionario de Lengua
L_2 : Palabras que empiezan por a
$L_3 : \{00, 001, 100\}$ leng con 3 palabras
L_4 : conj codigos fuente.

24 Febrer 05

- Palabra vacía: es aquella que no tiene símbolos (λ). → $|\lambda| = 0$
- longitud de la palabra: es el núm de símbolos que contiene ($|w|$).
- a-longitud de la palabra ($a \in \Sigma$): es el núm de a's que contiene ($|w|_a$).

Ejemplo:

$\Sigma = (0,1)$

$w_1 = 01001$	$\lvert w_1\rvert = 5$	$\lvert w_1\rvert_1 = 2$	$\lvert w_1\rvert_0 = 3$
$w_2 = 101$	$\lvert w_2\rvert = 3$	$\lvert w_2\rvert_1 = 2$	$\lvert w_2\rvert_0 = 1$
$w_3 = \lambda$	$\lvert w_3\rvert = 0$		

- subpalabra o factor: de una palabra es qualquier subcadena de símbolos consecutivos

Ej: $w_1 = 01001$ → subpalabras 01 001 λ és un submot de qualsevol mot.

- prefijo de una palabra es cualquier subcadena de símbolos al principio de la palabra.

Ej: $w_1 = 01001$ → 01, 010, 0100, 01001

- sufijo de una palabra es cualquier subcadena de símbolos al final de la palabra.

Ej: $w_1 = 01001$ → 1, 01, 001 ... 01001

- prefix, sufix propi de w → prefix/sufix que no pot ser ni λ ni w.
- Lenguaje universal de Σ, es el conjunto de todas las palabras posibles que se pueden construir a partir de Σ (Σ^*) (conjunt infinit)

$$\lambda \in \Sigma^* \quad \forall \Sigma$$

$L_1 = \{ w \in \{0,1\}^* \mid |w| = \dot{2} \}$ → Palabras de longitud par

$L_2 = \{ w \in \{0,1\}^* \mid |w|_1 = |w|_0 \}$

$L_3 = \{ w \in \{0,1\}^* \mid \text{Bin}(w) \text{ es primo} \}$
yuxtaposición de símbolos   núm natural

$L_4 = \{ w \in \text{ASCII}^* \mid$ son sintácticamente correctas en java. $\}$

$L_6 = \{ w \in \text{ASCII}^* \mid w \in L_4$ y no entran en bucle $\}$

Concatenación de palabras:

$$\Sigma^* \cdot \Sigma^* \to \Sigma^*$$

$(w_1, w_2) \to w_1 \underset{\text{es pot ometre}}{\cdot} w_2$ → w_2 a continuació de w_1
⇓
consisteix a juxtaposar els 2 mots

Ej: $\left.\begin{array}{l} w_1 = aab \\ w_2 = cca \end{array}\right\}$ $w_1 w_2 = aabcca$

* Propiedades concatenación:
  - Asociativa $(w_1 w_2) w_3 \equiv w_1 (w_2 w_3) \equiv w_1 w_2 w_3$
  - Element neutre λ
    $\lambda w = w \lambda = w \quad \forall w \in \Sigma^*$
  - NO commutativo: $w_1 w_2 \neq w_2 w_1 \quad \forall w_1 w_2 \in \Sigma^*$
    $w_1 w_2 = aabcca$
    $w_2 w_1 = ccaaab$

Redefinición prefijo:

x es prefijo de ω si $\exists y \in \Sigma^*$ tal que $\omega = x \cdot y$

Redefinición sufijo:

x es sufijo de ω si $\exists y \in \Sigma^*$ tal que $\omega = y \cdot x$

Redefinición factor:

z es factor de ω si $\exists x, y \in \Sigma^*$ tal que $\omega = x\ z \cdot y$

NOTA: λ es prefijo, sufijo y factor para cualquier palabra.

Σ^* es un lenguaje,
las palabras $\in$ a los
lenguajes, los lens
pueden o no estar
incluidos en otros
lenguajes

- *Potència de mots:*

$\omega^0 = \lambda$

$\omega^{i+1} = \omega^i \omega = \omega \omega^i \quad \forall i \geq 0$

$(ab]^3 = ababab$

$(ab)^0 = \lambda$

Concatenación de Lenguajes: Dados $L_1 . L_2 \subseteq \Sigma^*$ (lenguajes)

$L_1 L_2$ es el conjunto de todas las palabras que es posible obtener concatenando una palabra de L_1 con otra de L_2.

Formalmente: $L_1 L_2 = \{\omega \mid \exists x \in L_1 \wedge \exists y \in L_2 , \omega = x \cdot y\}$

$L_1 = \{b, ba, bba\}$
$L_2 = \{a, aa\}$

$L_1 L_2 = \{ba . baa, baaa, bbaa. bbaaa\}$
no repetidos. → és un conjunt

$L_1 = \{a, aa\}$
$L_2 = \{\lambda, a\}$

$L_1 L_2 = \{a, aa, aaa\}$

$L_1 = \{a, ab\}$
$L_2 = \{a . baa\}$

$L_1 L_2 = \{aa, abaa, aba. abbaa\}$

Podría passar que se perdieran elementos.

→ Propiedades:

- Asociativa: $(L_1 L_2) L_3 = L_1 (L_2 L_3) = L_1 L_2 L_3$
- Elemento Neutro: $\{\lambda\}$ lenguaje que solamente contiene λ

  $L \cdot \{\lambda\} = \{\lambda\} \cdot L = L$
- No commutativo:

  $L_1 = \phi$ lenguaje vacío $\Rightarrow |L| = 0 \not\Leftrightarrow L = \{\lambda\} \rightarrow \|L\| = 1$

  $L_2 = \{a, ab\}$ $\quad L_1 L_2 = \phi$

  $L \cdot \phi = \phi L = \phi$

$L = \{\omega \mid |\omega| = \dot{2}\}$ Palabras de longitud par

$L \cdot L = L$

Nota: Cal tenir en comple que el nostre llenguatge inclogui λ, perquè en el cas que no la contingués $L \cdot L = L$ no seria cert perquè una $|\omega| = 2$ només es podría formar per $\omega = \omega_1 \omega_2$ on $|\omega_1| = |\omega_2| = 1$. Altrament, si λ està inclosa $\omega_1 = \omega_2 \cdot \lambda$ on $|\omega_2| = 2$ i $|\lambda| = 0$.

⇒ Demostració formal:

Partint que $L \cdot L \subseteq L$

Sigui $\omega \in L \cdot L \Rightarrow \exists y, x \in L$, $|x| = \dot{2}$ i $|y| = \dot{2}$, $\omega = x \cdot y$

$\Rightarrow |\omega| = |x| + |y| = \dot{2}$ (la suma de 2 núm pares da un par)!

$(\underline{2})$

Sigui $\omega \in L \cdot L \Rightarrow \exists y, x \in L$ tal que $\omega = x \cdot y$

Basta considerar

$$\left.\begin{matrix} x = \lambda & |\lambda| = 0 = \dot{2} \\ y = \omega & |y| = \dot{2} \end{matrix}\right\} |\omega| = \dot{2}$$

$L \cdot \Sigma^* = \Sigma^*$

L contiene λ, todas las palabras de Σ^* concatenadas con λ, dan todas las palabras de Σ^*. El resto de palabras de L por definición de Σ^* ya están en Σ^*.

· Potències de Llenguatges :

$L^0 = \{\lambda\}$

$L^{i+1} = L \cdot L^i = L^i \cdot L \quad \forall i \geq 0$

· Complementario de un lenguaje: $\bar{L} = \Sigma^* - L$

↳ palabras de Σ^* que no están en L.

$\Rightarrow \quad L \cdot \bar{L} = \bar{L}$

($\subseteq$) $|w_1| + |w_2| \Rightarrow$ la longitud es impar $w_1 w_2 \in \bar{L}$
$w_1 \in L \quad w_2 \in \bar{L}$

($\supseteq$) Utilitzant λ

$L_1 = \{ w \in \{a,b\}^* \mid |w|_a = |w|_b \}$
$L_2 = \{ w \in \{a,b\}^* \mid |w|_a \geq |w|_b \}$
$L_3 = \{ w \in \{a,b\}^* \mid |w|_a \leq |w|_b \}$

1. $L_1 L_1 = L_1$
   - $\subseteq$ Dóna una paraula que conté $|w|_a = |w|_b$ $\quad w_1 \in L_1$, $w_2 \in L_1$
   - $\supseteq$ Es pot formar una paraula w tq. $|w|_a = |w|_b$ a partir de λ
2. $L_1 L_2 = L_2 \quad (L_1 \subset L_2)$
3. $L_1 L_3 = L_3 \quad (L_1 \subset L_3)$
4. $L_2 L_3 = L_3 L_2 = \{a,b\}^*$
   - $\supseteq \quad w \in \{a,b\}^*$ · Podem construir a partir de λ ja que λ està a $L_1 . L_3$
   - $|w|_a = |w|_b \Rightarrow w = w \cdot \lambda$
     - $w \in L_2$, $\lambda \in L_3$ o bé $w \in L_3$, $\lambda \in L_2$
   - $|w|_a \geq |w|_b \Rightarrow w = w \cdot \lambda$
     - $w \in L_2$, $\lambda \in L_3$ } + nombre d'a's
   - $|w|_a \leq |w|_b \Rightarrow$
     $w = \lambda \cdot w$
     - $w \in L_3$, $\lambda \in L_2$

$\Rightarrow$ si L_2 i L_3 són estrictes $<, >$

$L_2 L_3 = ?$ $\Rightarrow$ no és fàcil descriure'l
tenen un prefix que té + a's que b's
i un sufix amb + b's que a's.

· Reverso (revessat) de una palabra (w^R) es la palabra "leída" de derecha a izquierda.

Ej: $\quad w = aab \rightarrow w^R = baa$

Propiedades:

$$\begin{cases} (w^R)^R = w \\ \lambda^R = \lambda \\ (x \cdot y)^R = y^R \cdot x^R \end{cases}$$

· Reverso de un lenguaje (L^R): lenguaje formado por el reverso de las palabras de L.

$$\boxed{L^R = \{ w^R \mid w \in L \}}$$

Ej: $\quad L = \{ w \mid \exists y \; w = a \cdot y \}$
$\qquad L^R = \{ w \mid \exists y \; w = y \cdot a \}$

· Palíndromo (cap-i-cua): igual a su reverso $w = w^R$

Lenguaje asociado a un problema decisional

$L = \{ x \in \Sigma^* \mid x \in E \wedge \exists y \in S, P(x,y) \}$

$L = \{ x \in \Sigma^* \mid x \in E \wedge P(x) \}$

Resolver un problema decisional es absolutamente equivalente a decidir si una palabra pertenece a su lenguaje asociado.

25 Febrer05

Problema decisional

función asociada: Dado $x \in E$ calcular, determinar $y \in S$ tal que $P(x,y)$.

Dominio de una función:

$f: A \rightarrow B$ es el subconjunto más Grande de A tal que f está definida en él.

$f: \mathbb{N} \rightarrow \mathbb{R}$

$Dom(f) = \mathbb{N} - \{0\}$

$f(x) = 1/x$

selecciona y si hay más de 1a salida

$$f(x) = \begin{cases} y & \text{si } \exists y \ P(x,y) \\ \textcircled{\uparrow} & \text{sino} \end{cases}$$

Notación $f(x)^{\uparrow}$ ($f(x)$ no está definida)

$$f_L(x) = \begin{cases} 1 & \text{si } x \in L \\ 0 & \text{si } x \in \overline{L} \ (x \notin L) \end{cases}$$

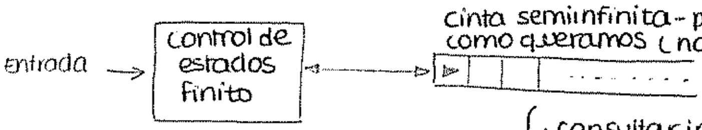
$\uparrow$
función característica de L

Un problema decisional es un caso particular de un probl funcional. L asociado a un prob. decisional

$\Rightarrow$ Resolent f_L resolem el probl decisional

• Tema 2: MÁQUINAS DE TURING.

Cada vez que definimos un algorismo detrás tenemos como mínimo una máq. de Turing para dicho alG. $\Rightarrow$ Máquina formal para un alG.



cinta semiinfinita - podemos almacenar tanta inform. como queramos (no limitada).

- Consultar info
- Modificar
- Desplazar

En cada celda colocamos un único símbolo de un alG $\Rightarrow$ Lo que hay en la cinta siempre será una palabra $\Rightarrow$ dado que por muy Grande que sea su contenido, siempre será finito (cinta semiinfinita)

Una máquina de Turing es una séxtupla $M = (\Sigma, \Gamma, Q, \delta, q_0, q_f)$.

Σ es el alfabeto de entrada (donde definimos las entradas) $\Rightarrow$ siempre necesitamos una entrada aunque sea vacía

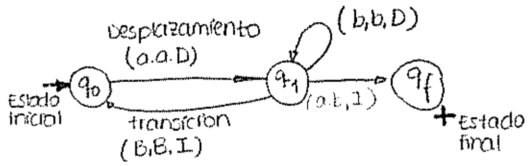
Γ (Gamma): Alfabeto de cinta $\triangleright, B \in \Gamma$ $\Sigma \subseteq \Gamma$. $\Rightarrow \triangleright, B \notin \Sigma$.

Q es un conjunto finito de estados.

δ (Delta): es la función de transición (nos dice realmente que realiza la máq.)

q_0: Estado inicial ($q_0 \in Q$)

q_f: Estado final ($q_f \in Q$). Estado al cual queremos llegar.



$Q = \{q_0, q_1, q_f\}$

derecho izquierdo Nada

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times (D, I, N)$

$(q_0, a) \rightarrow (q_1. a. D)$

$(q_1 \times a) \rightarrow (q_f. b. I)$

estoy viendo una a en la cinta

modif la a por una b

El conj de todas las transiciones forma la función de transición $\rightarrow \delta$

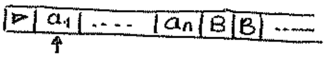
* situación inicial :

  - Empezamos en q_0.
  - En la cinta tenemos la palabra de entrada.

    En Γ hay 2 caracteres especiales : ▶, B (blanco) que nos delimitan donde empieza y termina la inform.

    $\omega = a_1 \dots a_n$

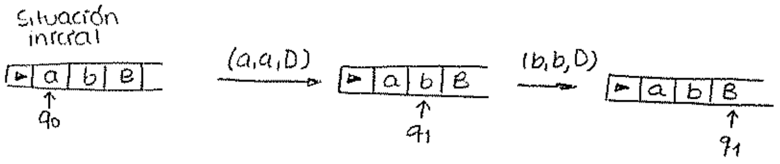
    cinta :



  - El cabezal de la cinta "apunta" a la segunda posición (En algunos libros, apunta al inicio)

Ejemplos :

* Suponemos que tenemos como entrada ab → ω = ab

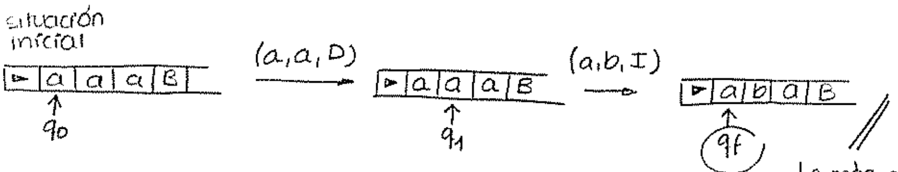


✓ siempre que tenga una transacción definida para la situación en que se encuentra la ejecuta

Aquí la maq. s'atura perquè arribem en un estat on el capçal llegeix b i la única transició definida en l'estat es per $\underline{\underline{a}}$ $(\underline{a}, a, D)$

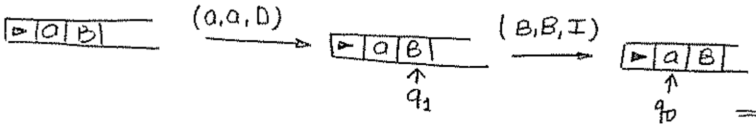
> * Nunca hay ambigüedad (nunca 2 trans def)
> * La máquina para cuando llega a una config para la que no tiene transacción definida
>
> Restricción : El estado final (q_f) $\underline{\underline{\text{no}}}$ tiene transacción definida

* ω = aaa



La máq para ⇒ desde qf no hi han transaccions definides (Restricció)

* ω = a

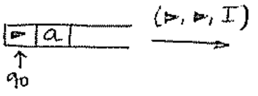


Estamos en un bucle, ⇒ Dada esta entrada la maq no para

Máquina con entrada x :

$M(x) \downarrow \Longrightarrow$ Para

$M(x) \uparrow \Longrightarrow$ $\underline{\text{NO}}$ para



la máq. para (no error) !!

Diremos que una palabra $w \in \sum^*$ es aceptada/reconocida por una máquina de Turing M si $M(x)\downarrow$ en qf.

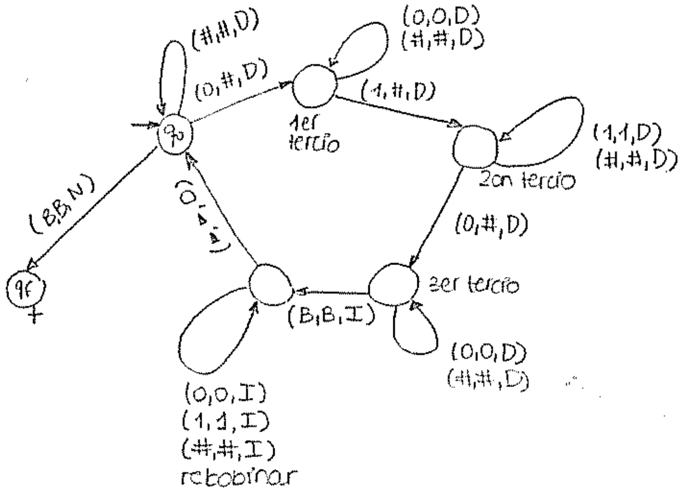
Al conjunto de todas las palabras aceptadas por una máq. Turing lo llamaremos lenguaje aceptado por una máq. Turing.

Ejemplo: construir una máq. de Turing que reconozca $L = \{0^n 1^n 0^n \mid n \geq 0\}$

* Maq. Turing que para las palabras de este lenguaje pare y lo haga en estado final.

$L = \{\lambda, 010, 001100, 000111000 \ldots\}$

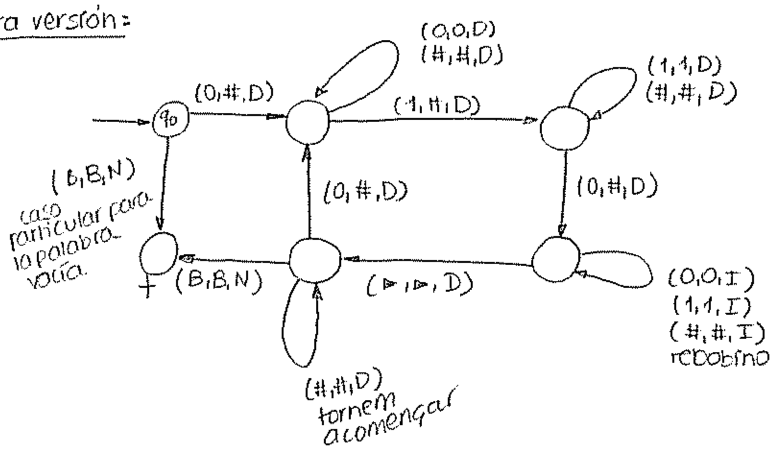
000 111 000
#00 #11 #00) rebobinar
##0 ##1 ##0)
Al final tot queda #



- $w = 0110$ ↑ ##1# ↑ esperamos encontrar un 0 ⇒ no hay trans def. ⇒ paramos

- $w = 0101$ ###1 ↑ no hay 0's ni blancos ⇒ llem marcat però esperem el final de la paraula.

⟹ * Otra versión:



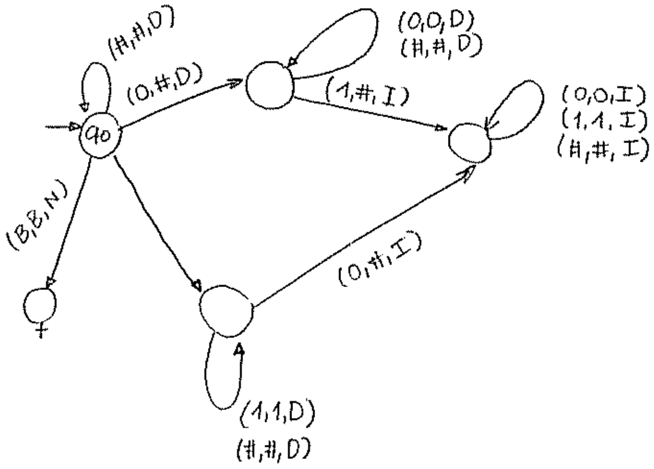
¡Falla! ⟹ reconoce la concatenación de palabras $0^n1^n0^n + 0^n1^n0^n + \ldots$

ex: $\underbrace{010}\underbrace{010}$

$\boxed{L^*}$ conj de todas las palabras concatenando k veces las palabras de L.

* Ejercicio: construir una máq. de Turing que reconozca $L = \{w \mid |w|_0 = |w|_1\}$ $\Sigma = \{0,1\}$

Estrategia: ir marcando 0's i 1's por parejas

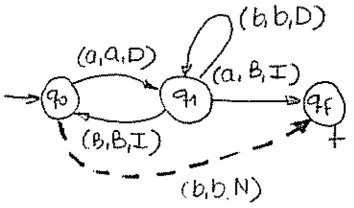


001 00011 001111
# #
#
#

28 Febrero05

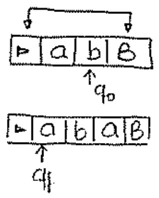
- $L(M) = \{w \in \Sigma^* \mid M(w)$ para en estado final$\}$

- La salida de una TM con entrada w: M(w) es lo que hay en la cinta entre los dos primeros símbolos que no son de Σ. ← alfabeto de entrada



si $w = a$ $M(w)\uparrow$
$w = ab$ $M(w)\downarrow$ → para en q_0
$w = aaa$ $M(w)\downarrow$ → para en q_f

máq. entra en bucle



M(ab) = ab

M(aaa) = aba

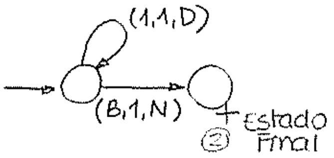
- La función calculada por una MT es:

$$f_M(w) = \begin{cases} M(w) & \text{sí } M(w)\downarrow \\ ④ \text{ indefinida} & \text{sí } M(w)\uparrow \end{cases}$$

Podemos no permitir parar en un estado NO final
Definir todas las transiciones que faltan para hacer que todos los estados vayan a parar siempre a qf.
El qf, solamente interesa en probl decisionales *

⇒* Ejemplo: construir una TM para tener la función $f(1^n) = 1^{n+1}$ $n \geq 0$; $\Sigma = \{1\}$

11B → 111 $\boxed{1^0 = \lambda \text{ Palabra vacía}}$



Para calcular funciones no nos importa si para o no en estado final ⇒ ya que parar en estado final solo sirve para reconocer lenguajes.

$L(M)_{①} = \emptyset$
$L(M)_{②} = \Sigma^*$ (El conj de todas las palabras que podemos construir con $\Sigma = \{1\}$)

- una TM de <u>parada segura</u> es aquella que para con cualquier entrada $w \in \Sigma^*$.

Dada una TM de parada segura M; se define $T(M,w)$ como el núm de transiciones que necessita M para parar con entrada w.

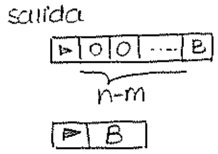
El <u>tiempo de cálculo</u> de una TM de parada segura se define como una función de la longitud de la entrada:
$$\boxed{t_M(n) = \max_{|w|=n} T(M,w)}$$
⟹ caso Peor

El tiempo lineal → $t_M(n) = O(n)$ $\exists k : t_M(n) \le k \cdot n$
tiempo polinómico → $t_M(n) \Rightarrow \exists p(n) : t_M(n) = O(p(n))$
tiempo exponencial → $t_M(n) = O(2^n)$

* Ejemplo: construir una TM que calcule:

$$f(0^n \# 0^m) = \begin{cases} 0^{n-m} & \text{sí } n>m \\ \lambda & \text{si no} \end{cases} \quad ; \ \Sigma = \{0,\#\}$$

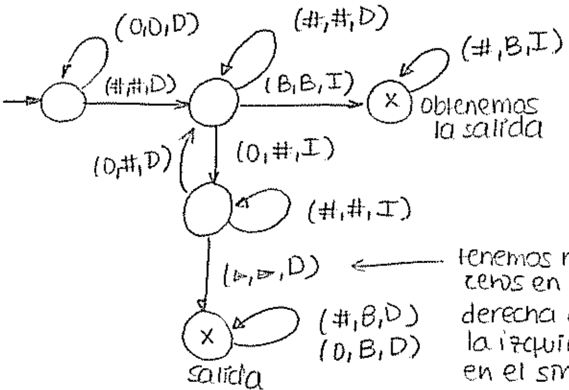
la entrada siempre es válida ⟹ No hace falta verificar el formato de la entrada



$n,m \ge 0$

⟹ Estrategias:

▷ 000000# 00000 B
▷ X00000# 0000X B
▷ X00000# X0000 B
▷ 0000X #0000X B
▷00000## #0000 B



tenemos más ceros en la parte derecha que en la izquierda estamos en el sino ⟹ λ
Bastaría con poner un B en la primera posición ⟹ no haría falta limpiar toda la cinta.

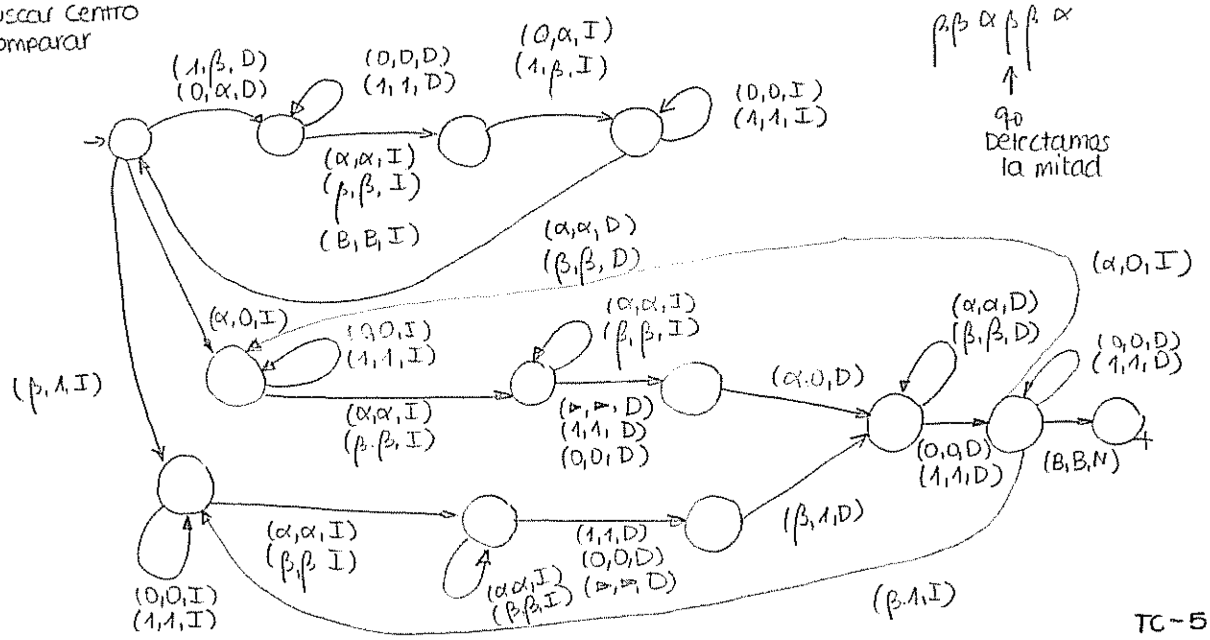
Estamos calculando funciones, no nos importa el estado final.

⟹ Ejercicio 1.3:
* construir una TM que reconozca $L = \{ww \mid w \in \{0,1\}^*\}$

(1) Buscar Centro
(2) Comparar

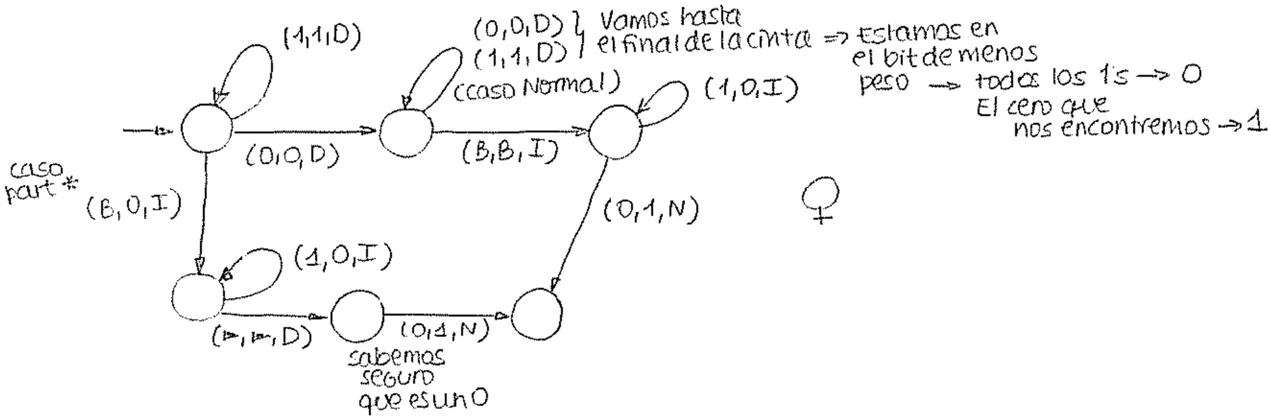
* Detrás versión prof ①*

1 1 0 1 1 0
β β α β β α
↑
q_0
Detectamos la mitad



* construir una TM que calcule la función $f(x) = x+1$ $x \in \{0,1\}^*$

si acaba en 0 → ···· 0 ↓ ···· 1	
si acaba en 1 → 0111 1000	
si todo son 1's → ⊢111B suma→1000	✓ caso particular *
0 1	



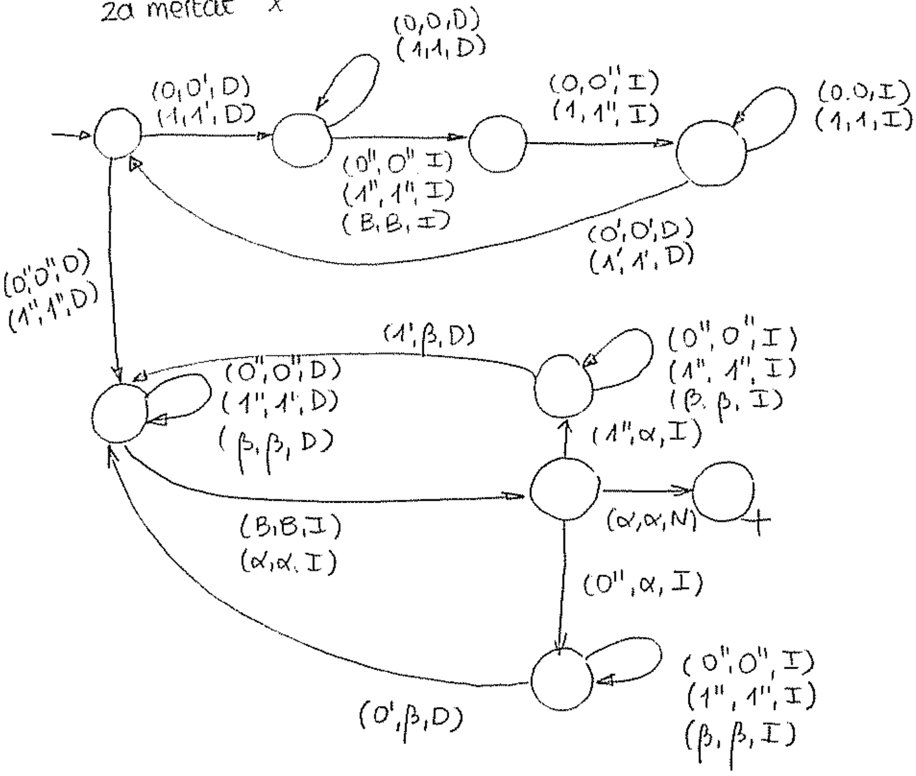
} Vamos hasta el final de la cinta ⇒ Estamos en el bit de menos peso → todos los 1's → 0
El cero que nos encontremos → 1

sabemos seguro que es un 0

---

① * Versión Prof:

1a meitat x'
2a meitat x''



0' 1' 1' 0'' 1'' 1''
β β β α α α

- Diremos que un lenguaje es <u>Enumerable Recursivamente</u> si existe una TM que lo reconoce:

$$L \text{ es enumerable recursivamente} \iff \exists_M \ \mathcal{L}(M) = L$$

- Diremos que un lenguaje es <u>decidible</u> si existe una TM de <u>parada segura</u> que lo reconozca:

$$L \text{ es decidible} \iff \exists M \ \mathcal{L}(M) = L \wedge \forall w \in \Sigma^* \ \boxed{M(w)\downarrow}$$

Ejemplos:

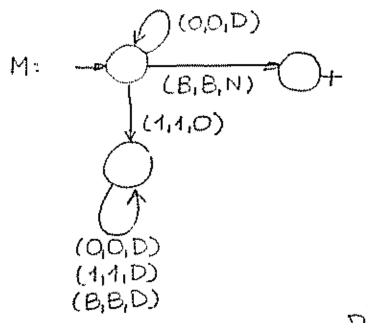
$L_1 = \{0^n 1^n 0^n \mid n \geq 0\} \rightarrow L_1$ es decidible y enumerable recursivamente

$L_2 = \{ww \mid w \in \{0,1\}^*\} \rightarrow L_2$ es decidible y enumerable recursivamente.

Propiedad: Decidible ⇒ Enumerable Recursivamente

¿Qué lenguaje reconoce esta TM?

M:



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

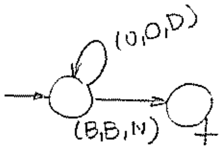
$\mathcal{L}(M) = \{0^n \mid n \geq 0\}$

→ M <u>no</u> es de parada segura
siempre que contenga un 1
entra en bucle.

→ Es enumerable recursivamente
para $\mathcal{L}(M)$

Esta maq no es decidible pero se
puede encontrar alguna que reconozca
el mismo lenguaje y lo sea ⇒ M'

⇒ M':



- Diremos que una <u>función es calculable</u> si existe una TM que la calcule:

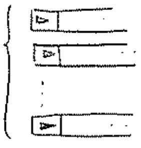
$$f \text{ es calculable} \iff \exists M \ f_M = f$$

⇒ Extensiones del modelo básico de Máquina de Turing

- TM Multicinta:

Tenemos K cintas semiinfinitas

K



$\overbrace{a_1, b_1, m_1}^{\text{cinta } 1}; a_2 . b_2 . m_2; \ldots ; \overbrace{a_K b_K m_K}^{\text{cinta } K})$

configuración inicial

cinta entrada: ▶| w | B

K) ▶| B | ···

⋮

cinta salida: ▶| B | ···

Teorema: Las TM multicinta tienen la misma potencia, $\boxed{\text{capacidad de cálculo}}$ que las TM con una cinta.

↓

si L es reconocido por una TM multicinta, entonces existe una TM con una sola cinta que también reconoce L.

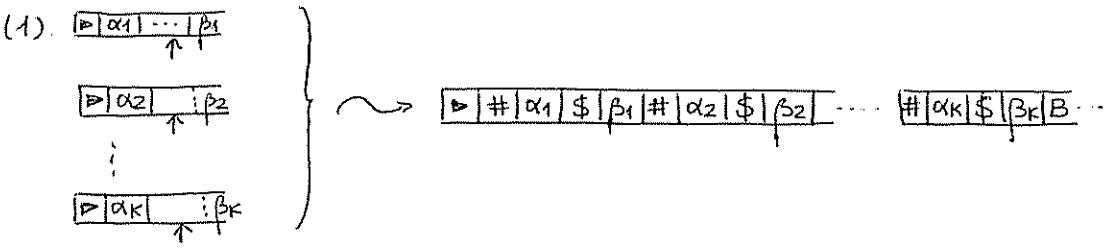
si f es una función calculada por una TM multicinta, entonces existe una TM con una sola cinta que también calcula f.

* Demostración (esbozo) : $\boxed{\text{TM multicinta} \subseteq \text{TM unicinta}}$

Idea: Poner en la única cinta que tengo, todas las cintas =>

1) Guardar el contenido de todas las cintas en una
2) Para cada transición 'multicinta', hacer la modif correspondiente en cada 'cinta simulada'.

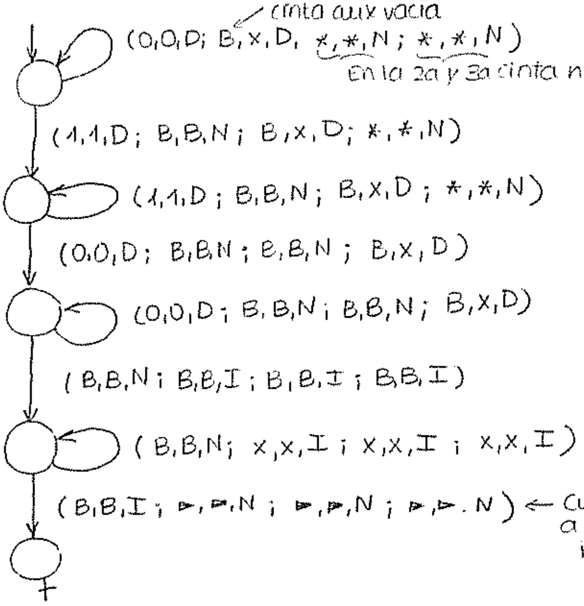
(1)



Ejemplo multicinta : $L = \{0^n 1^n 0^n \mid n \geq 0\}$

Guardar 1a cinta 0's → recorrer-la toda
2a cinta 1's → mirar si terminamos en el mismo punto
3a cinta 0's

cinta aux vacia

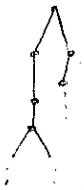


En la 2a y 3a cinta no hacemos nada.

← Cuando llegamos a la vez al simbolo inicial.

otras variaciones :

Indeterminismo - se va generando un árbol donde se van generando las distintas configuraciones



A partir de una cierta conf, tenemos bifurcaciones ..

Guardar en la cinta el recorrido en anchura del árbol

varios cabezales

cinta infinita bidireccional → por los dos lados

04 Marzo 05

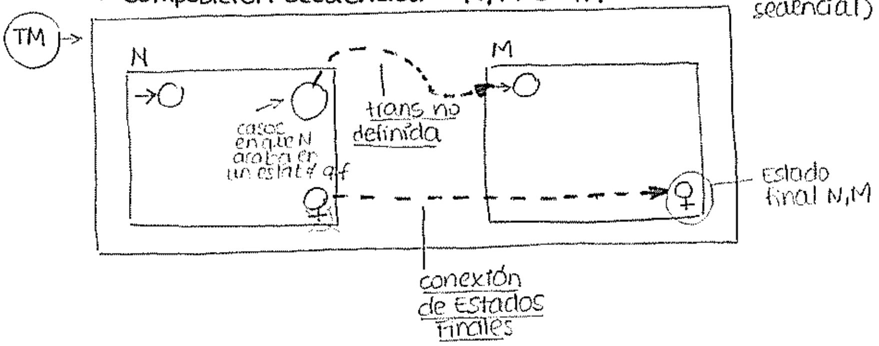
• Tema 3. Máquinas de Turing y Algoritmos

- Esquemas algorítmicos básicos < composición secuencial / composición condicional / composición iterativa.
- Codificación de las TM → palabra sobre un cierto alfabeto.
- Intérpretes y simuladores → Máq. Universal.

⇒ Tesís de CHORCH-TURING (Es una hipótesis)
Toda la teoría de la computación se basa en esta hipotesis.

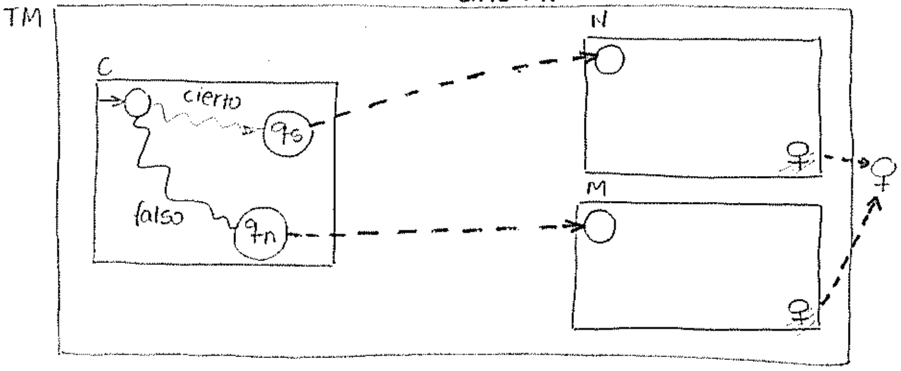
3.1. Esquemas algorítmicos básicos

* composición secuencial N, M ∈ TM N,M (Máq. secuencial)

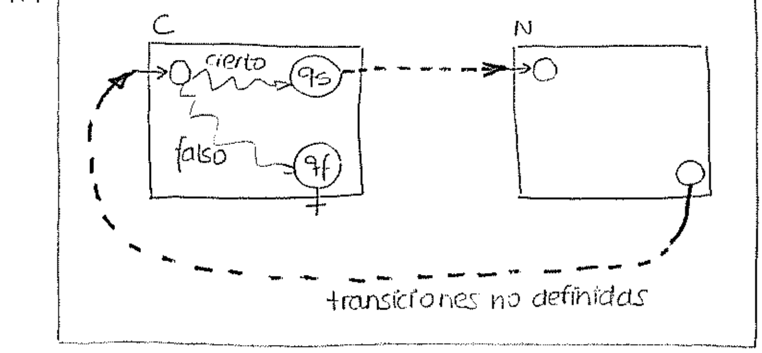


⇒ construir una máq que ejecute las 2 de forma secuencial

* composición condicional : C, N, M ∈ TM.
Si C entonces N
sino M.



* composición iterativa : mientras C hacer
N



3.2. Codificación de las Máquinas de Turing

Codificar: Procedimiento que nos permite convertir un objeto en otro (Natural ⇒ binario).
Necesitamos que sea finita.

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

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times (D, I, N)$$

Posible codificación:

- los estados se numeran del 0 al n
- $q_0 \rightarrow 0$, $q_f \rightarrow n$
- los elementos de Σ y Γ los ordeno

  $\Sigma : \sigma_1, \sigma_2, \ldots, \sigma_m$

  $\Gamma : \gamma_1, \gamma_2, \ldots, \gamma_r$

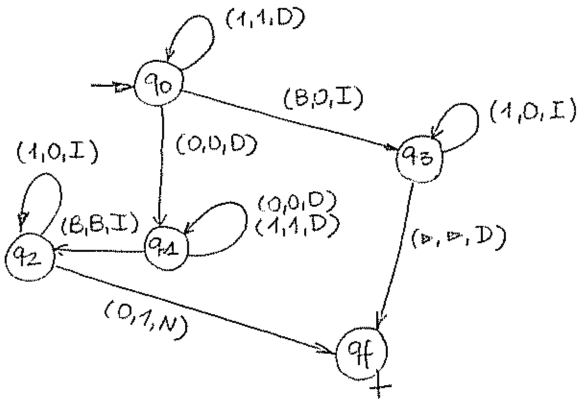
- (D,I,N) : movimientos

  D → 01
  I → 10
  N → 00

- $\delta(q_{i1}, q_{j1}) = (q_{i2}, \gamma_{j2}, m_{ij})$ se codifica como: $q_{i1} \cdot \gamma_{j1} \cdot q_{i2} \cdot \gamma_{j2} \cdot m_{ij}$
- La codificación podría ser:

$$\text{núm. estados} \overset{\swarrow \text{caract separador}}{\#} \text{núm. elems } \Sigma \# \text{núm elems } \Gamma \underbrace{\underbrace{\# i_1 \# j_1 \# i_2 \# j_2 \# m_{ij}}_{\text{una transición}} \# \underbrace{\ldots}_{\text{trans}} \# \underbrace{\ldots}_{\text{trans}}}_{\text{definimos todas las trans}} \ldots$$

Ejemplo:



núm. estados = 4

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

$\Gamma = \{0, 1, B, \triangleright\}$

Función de transición: ⟶

Q	Γ	Q	Γ	m
q_0	1	q_0	1	D
	0	q_1	0	D
	B	q_3	0	I
q_1	0	q_1	0	D
	1	q_1	1	D
	B	q_2	B	I
q_2	1	q_2	0	I
	0	q_4	1	N
q_3	1	q_3	0	I
	▷	q_4	▷	D

codificación:

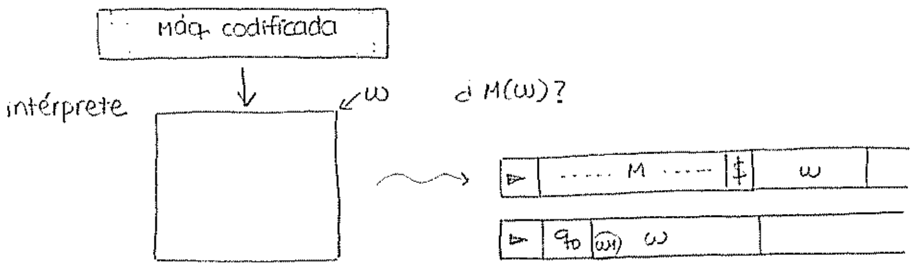
$$5\#2\#4\#\underbrace{0\#1\#0\#1\#01}^{\text{1a transición}}\#\overbrace{0\#0\#1\#0\#01}\#\cdot$$
$$\#\ 0\#B\#3\#0\#10\#1\#0\#1\#0\#01\#\cdot$$
$$\vdots$$
$$\#\ 3\#\triangleright\#4\#\triangleright\#01\#$$

Esta codificación satisface:

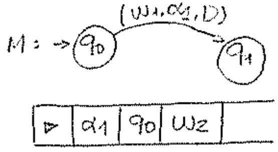
1. Es una palabra → hay un alfabeto detrás y una yuxtaposición de símbolos.
2. Existe un mecanismo para verificar si una cierta palabra (presunta codificación) es realmente una codif. válida.
3. No es ambigua: codificaciones diferentes vienen de TM diferentes. TM diferentes dan codificaciones diferentes.
4. Se puede pasar todo a binario ($\Sigma = \{0,1\}$) sin perder ninguna propiedad → 16 símbolos en este caso.

3.3. Intérpretes y simuladores

Notación:
M_x → TM cuya conf es x
φ_x → función calculada por M_x
L_x → lenguaje aceptado por M_x.
$<y>$ → "y codificada"



Buscar en la tabla de transiciones la pareja (q_0, w_1)



El estado que llegará el intérprete lo sabrá siempre.

Máquina Universal: (Hay ∞ máq. univ)

entrada $<x,w>$

x → codificación de una TM.
ω → palabra de entrada.

```
[f := configuración_inicial(Mx, ω);]  M1
mientras [f no es conf. terminal en Mx]  M2
    [f := siguiente_configuración(Mx, f);]  M3
fmientras
escribir f ...
```

⇒ (M1 mientras M2 M3) ∈ TM

- Máquina universal con reloj: → simular t pasos de la máq Mx con entrada ω

```
entrada <x,ω,t>   <- # passos a fer
f := configuración_inicial(Mx,ω);
i := 0;
mientras (i<t ∧ f no es terminal para Mx) hacer
        f := siguiente_configuración(Mx,f);
        i++;
fmientras
{ escribir(f,i)
{ ó
{ escribir(f, 1ó0)  -> 1 => Mx↓ en menos de t pasos
                    -> 0 => Mx↑
```

↓
(1) Es una TM
(2) Es de parada segura (la máq univ no es de parada segura)

Las TM:

- son capaces de implementar esquemas algorítmicos básicos.
- Tienen codificación → pueden ser entrada de otras TM.
- Existen TM universales que interpretan a cualquier TM.

⇒ Tesis de Church-Turing: La noción intuitiva de algoritmo se corresponde con la definición formal de la TM.

Algunos problemas sobre TM:

- Problema de Pertenencia: $PERT = \{ <x,\omega> \mid \omega \in \mathcal{L}(M_x) \} = \{ <\omega,x> \mid M_x(\omega)\downarrow \text{ en } q_f \}$
- $HALT = \{ <x,\omega> \mid M_x(\omega)\downarrow \}$
- Equivalencia en cuanto al lenguaje reconocido: $EQUIV = \{ <x,y> \mid \mathcal{L}(M_x) = \mathcal{L}(M_y) \}$
- Problema K: $K = \{ <x> \mid M_x(x)\downarrow \}$ ← ¿la máq con entrada x, para?
- $DOM = \{ <x> \mid \mathcal{L}(M_x) \neq 0 \}$ máq. cuyo lenguaje no es vacío → aunque sea λ

7 Març05.

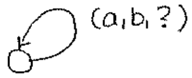
Problema 2.1:

TM inquieta: en cada transición se mueve el cabezal (E,D).

Demostrar que el conjunto de lenguajes reconocidos por TM inquietas es el mismo que por TM.

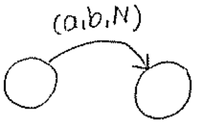
- Dada una TM inquieta ⇒ ∃ TM "normal" que reconoce el mismo lenguaje.
  La misma máq. ⇒ Está dentro la misma definición de TM
- Dada una TM "normal" ⇒ ∃ TM inquieta.
  si en una TM normal pasa:

Podemos fusionar estados :

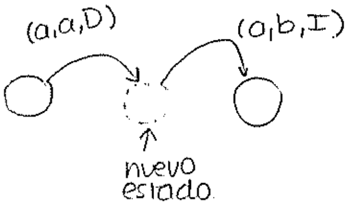


Esto no contempla todos los casos

si tenemos una TM normal tenemos transiciones que < mueven el cabezal / no mueven el cabezal



Creamos un nuevo estado que primero hace a la D y luego a la I



si hacemos esta operación en todas las transiciones que no hacen nada, hemos convertido la TM "normal" en una TM inquieta ⇒ Hemos demostrado que dada una TM normal existe una TM inquieta que reconoce el mismo lenguaje

Problema 3.1:

Encontrar una codificación en binario para las palabras $w \in \Sigma^*$, Σ cualquiera

- No es ambigua
- Ser posible encontrar el obj a partir de la codificación y viceversa
- Finita.

1) $\Sigma = \{a_1, \ldots, a_n\}$

Codificación unària:

$$\begin{cases} 0 \rightarrow \text{separador de símbolos} \\ \Longrightarrow a_k = 1^k \end{cases}$$

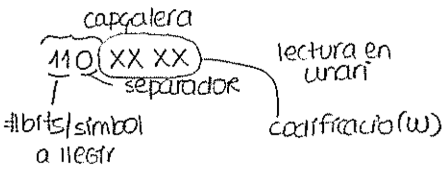
$w = w_1, \ldots, w_m = a_{i1} . a_{i2} \ldots a_{im}$

Codificación $(w) = 1^{i_1} 0 1^{i_2} 0 \ldots 0 1^{im}$

2) Dependiendo del # bits necesarios para representar una long finita ⟶ $\boxed{\# \text{bits} = \log_2 n}$

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

a = 00
b = 01
c = 10



✓ No toda combinación binaria es válida.
p.ex.: 110 11
no válido !!

3) Assignar a cada w de Σ^* un núm. natural que es la posición que ocupa en el orden léxico de Σ^* (el cual está ordenado).

orden lexicográfico ordenado → para $a_1, a_2 \mid a_1 \neq a_2$ y $|a_1| < |a_2| \Rightarrow a_1$ va antes que a_2.

si son = ⇒ en orden alfabético

Ex: $\Sigma = \{a, b, c\}$
$\{\overset{1}{\lambda}, \overset{2}{a}, \overset{3}{b}, \overset{4}{c}, \overset{5}{aa}, \overset{6}{ab}, \overset{7}{ac}, \overset{2}{ba}, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, \ldots$

← l'ordre cal transformar-lo a binari

Sea M una TM, $w \in \Sigma^*$

estado y contenido de la cinta

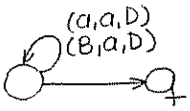
1 Si M con entrada w no repite configuración, entonces $M(w)\downarrow$

2 Si $M(w)\downarrow$, entonces no repite ninguna configuración

3. Si M con entrada w, no repite configuración, entonces $M(w)\uparrow$

Respuestas:

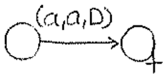
1. Falso. • Contraexemple: $w=a$



Veiem que sempre tira a la dreta i no acaba mai, mai es repeteix conf ja que sempre hi ha un element 'B' que passa a 'a' a cada pas de computació.

2. Cierta. Porque si repitiera conf no pararía => entramos en bucle

3. Falso.



10 Març 05

• Tema 4 : Computabilidad y decidibilidad

f : función computable → Existe TM que la calcule

L enumerable recursivamente → Existe TM tal que $\mathcal{L}(M) = L$ ← Trobar una màq. que reconegui un llenguatge

L decidible → Existe TM de parada segura tal que $\mathcal{L}(M) = L$

$\boxed{\text{L decidible} \Rightarrow \text{L enumerable recursivamente (E.R)}}$

Ejemplos:

$L = \{0^n 1^n 0^n \mid n \geq 0\}$ es decidible

$L = \{p \mid p \text{ es primo}\}$ es decidible

· $PERT = \{(x, \omega) \mid \omega \in \mathcal{L}(M_x)\}$ es E.R :

TM {
- Entrada $\langle x, \omega \rangle$
- simular Mx con entrada ω → Maq. Universal
- si la configuración contiene estado final → Aceptar (significa trans a qf)
- sino → Rechazar
  (significa transición a estado NO final que no tiene trans definidas)

· $HALT = \{(x, \omega\rangle \mid M_x(\omega)\downarrow\}$ es E.R :

TM {
- Entrada $\langle x, \omega \rangle$
- simular Mx con entrada ω
- $\boxed{\text{Aceptar}}$ (llegar las máq.* que no entran en bucle al simular con ω)
  (No importa en que estado pare, mientras lo haga)

· $K = \{\langle x \rangle \mid M_x(x)\downarrow\}$ es E.R : → No es decidible

TM {
- Entrada $\langle x \rangle$
- Simular Mx con entrada x
- Aceptar

· $T\text{-}HALT = \{\langle x, \omega, \overset{\text{tiempo máx.}}{t} \rangle \mid M_x(\omega)\downarrow \text{ en } \leq t \text{ pasos}\}$ es E.R

- Entrada $\langle x, \omega, t \rangle$
- Simular Mx con entrada ω, t pasos → Máq. Universal con reloj.
- si la configuración a la que llega Mx con entrada ω después de t pasos es terminal → Aceptar
- sino Rechazar (no hay transiciones que la hagan avanzar)

* Aquesta màq. és de parada segura, ja que la màq. univ de rellotge no és ⇓ el llenG. és E.R i decidible.

↳ Tot i que no sempre implica que si hi ha rellotge pari

EX :

```
t = 1
mientras la conf no es terminal
        simular Mx con t pasos;
        t++;
fmientras
Aceptar
```

← si no existe ninguna conf terminal → la MÁQ entra en BUCLE! ⇒ EX lenG K

• $DOM = \{\langle x \rangle \mid \mathcal{L}(M_x) \neq \phi\} \equiv$ Existe ω, $\omega \in \mathcal{L}(M_x) \Leftrightarrow \exists \overset{\Sigma^*}{\omega} \mid M_x(\omega)\downarrow$ acepta.

Aquí no nos dan la máq. ⇒ problema de Búsqueda → Σ^* está ordenado ⇒ ir provando todas

```
Entrada <x>
salir = falso;
ω = λ
mientras no salir
    si Mx acepta ω → salir = cierto;      (Maq univ)
    sino ω = ω+1 (o bien ω = siguiente(ω))
fmientras
⇒ Aceptar
```

Nadie nos garantiza que la encontramos pq. si λ ya entra en bucle ⇒ nunca se reconocerá DOM

⇒ Buscaremos cada vez más palabras en más tiempo → Buscar en un núm. pasos ⇒ **Máq. Reloj**

1 palabra → 1 paso
2 pal → 2 pasos
3 pal → 3 pasos
⋮ ⋮

TM
```
Entrada <x>
salir = falso
t = 1
mientras no salir
    para ω desde λ hasta t        ← la que ocupa la posición t.
        si Mx acepta ω en t pasos → salir = cierto fsi
                               ↑
                               máq
                               reloj universal
    fpara
    t := t+1
fmientras
Aceptar
```

- $\langle x,\omega\rangle \in HALT \overset{\text{k com}}{\to} M(\langle x,\omega\rangle)$
  $\langle x,\omega\rangle \notin HALT \to M(\langle x,\omega\rangle)\uparrow$

$\langle x,\omega\rangle \in PERT \Rightarrow M(\langle x,\omega\rangle)$ Acepta
$\langle x,\omega\rangle \notin PERT \Rightarrow M(\langle x,\omega\rangle)$ Rechazar / $\uparrow$

$\langle x,\omega,t\rangle \in T\text{-}HALT \to M(\langle x,\omega\rangle)$ Acepta
$\langle x,\omega,t\rangle \notin T\text{-}HALT \to M(\langle x,\omega\rangle)$ Rechaza

⇒ Demostrar que K es NO-DECIDIBLE (sabiendo que es ∈ R) :

$\boxed{\text{Teorema : K no es decidible}}$

* Proposición: La función $f(x) = \begin{cases} 1 & \text{si } M_x(x)\uparrow \\ \uparrow & \text{si no} \end{cases}$ NO es calculable.

No podemos calcular ninguna función que nos devuelva 1 si una máq NO para.

⇒) Demostración por Reducción al ABSURDO :

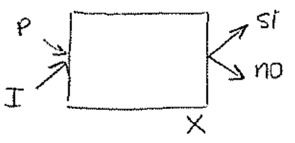
- Suponcamos que existe una TM M tal que M calcula f.
  sea x_0 la codificación de M ⇒ $\varphi_{x_0} = f$ → función calculada por M_{x_0} ≡ $\varphi_{x_0}(x) = \begin{cases} 1 & \text{si } M_x(x)\uparrow \\ \uparrow & \text{si no} \end{cases}$

  máq ↓ entrada ↓
  $\underbrace{\varphi_{x_0}(x_0)}_{f(x_0)}$
  a) $\xrightarrow{(a)}$ ① $= \varphi_{x_0}(x_0)$ ⇒ $M_{x_0}(x_0)\uparrow$!! Contradicción !!
  ↑ aplicamos la definición de f
  b) $\xrightarrow{b}$ ↑ $= \varphi_{x_0}(x_0)$ $\underset{\text{por def}}{\Rightarrow}$ $M_{x_0}(x_0)\downarrow$ contradicción
  Simulamos la máq. y para ⇒ res 1
  Aplicamos la def. f y vemos que no para !

simular ⇒ parar la carga

- Suponcamos K decidible ⇒ ∃M de parada segura t.q. $\overbrace{\mathcal{L}(M_K) = K}^{\text{reconoce K}}$
  sea M' :
  ```
  entrada <x>
  simular Mk con entrada x  --> es de parada segura pq suponemos que es decidible.
  si la configuración es aceptadora → bucle
  sino escribir 1.
  ```
  M' calcula f → contradicción pq en el paso anterior hemos demostrado que f NO es calculable.

- Problema1: Diseñar un programa que diga si otro programa P con entrada I escribe 'Hello world' o NO.

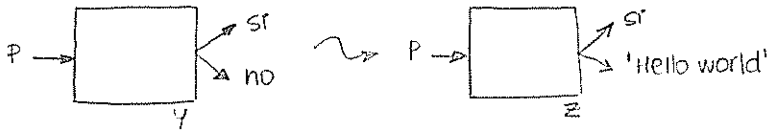


Supongamos que X existe ⇒ podríamos construir Y del problema 2

Y :
```
Entrada P
y := Ejecutar X(P,P)
Escribir y
```

Acabamos de diseñar una máq. que acabamos de demostrar que no existe ⇒ X NO EXISTE!

- **Problema 2:** Diseñar un programa Y que diga si un cierto programa P con entrada P escribe 'Hello world'.



Supongamos que z existe.

z con entrada z
- sí ⇒ z con entrada z escribe 'Hello world' ⇒ contradicción
  - pero si escribe Hello world es que no
- 'Hello world' ⇒ z con entrada z <u>no</u> escribe 'Hello world'.

⇒ z <u>NO</u> existe !!

X 11 Març 05

Demostración: HALT no es decidible: $HALT = \{<x,w> \mid M_x(w)\downarrow\}$

⇒ Reducción al absurdo

Supongamos que HALT es decidible: $\exists M_{halt}$ de parada segura acepta HALT.

Sea la máquina:

M
máq
de K
- Entrada $<x>$
- simular M_{HALT} con entrada $<x,x>$
- sí acepta → ACEPTAR
- sino → Rechazar

1. Es una máquina.
2. De parada segura (por hipótesis, si suponemos que HALT lo es M también)
3. $\mathcal{L}(M) = K$ → solo acepta si $M_x(x)\downarrow$ → como el lenguaje K

contradicción! Porque en la demostración anterior hemos obtenido que K no es decidible ⇒ Esta máquina M no existe!

⇒ $\boxed{\text{Halt no es DecidiBlE!}}$

Teorema: Sean A, B enumerables recursivamente. Entonces,

(a) $A \cup B$ es ER
(b) $A \cap B$ es ER
→ si A y B además son decidibles:
(c) $A \cup B$ es decidible
(d) $A \cap B$ es decidible
(e) $\overline{A}, \overline{B}$ es decidible
con respecto a Σ^*

Demostración: $A \cup B$ es E.R $\equiv \exists M$ tq $\mathcal{L}(M) = A \cup B$

$\exists$ <u>M_A, M_B</u> tal que $\mathcal{L}(M_A) = A$ y $\mathcal{L}(M_B) = B$ (Demostración constructiva).
no sabemos
si son de parada segura

```
Entrada x
t:=1; parar=false;
mientras no parar
    y:= simular t pasos de M_A con entrada x
    z:= simular t pasos de M_B con entrada x

    si  y es conf_final o z es conf_final → parar:=cierto;
    sino  t:=t+1;
fmientras
Aceptar
```

✓ no es queden penjades perquè simulen t passos ⇒ segur que la simulació acaba.

$M_{A\cup B}(x)$ → Aceptar $x \in A \cup B$
→ ↑ sino

Demostración: $A \cap B$ es E.R $\equiv \exists M$ tq $\mathcal{L}(M) = A \cap B$ ⇒ Canviar <u>o</u> por <u>i</u>

== Demostración : $A \cup B$ es decidible

$\exists M_A, M_B$ $\alpha(M_A) = A$ y $\alpha(M_B) = B$ de parada segura.

```
Entrada x
y:= simular M_A con entrada x
z:= simular M_B con entrada x
si y es conf_final ∨ z es conf_final => Aceptar
sino rechazar
```

↓ Es de parada segura porque M_A, M_B lo son.

== Demostración : $\bar{A}, \bar{B}$ son decidibles.

$\bar{A}$: Palabras de Σ^* que no son de A
$\exists M_A$ tal que $\alpha(M_A) = A$ de parada segura.

```
Entrada x
y := simular M_A con entrada x.
si y es conf_final → rechazar   // y∈A
sino aceptar   // y∈Ā
```

• Teorema: un lenguaje L es decidible si y solo si, tanto si L como $\bar{L}$ son enumerable recursivamente.
del complementari

=>) Evidente (todo decidible es E R).

<=) M_L $M_{\bar{L}}$

$L \cup \bar{L} = \Sigma^*$ nos basamos en la máquina construida anteriormente para $A \cup B$:

```
M   Entrada x
    t := 1
    salir := falso
    mientras no salir
        y:= simular t pasos de M_L con entrada x
        z:= simular t pasos de M_L̄ con entrada x

        si y es conf_final → salir = cierto, a = cierto
        si z es conf_final → salir = cierto, a = falso.

        t := t+1;
    fmientras
    si a → Aceptar      // xq aquesta màq reconeix L
    sino Rechazar
```

$\alpha(M) = L$
M es de parada segura. => $\boxed{\text{L es decidible !}}$

Teorema:

⟹ $\boxed{\text{Cualquier lenguaje E.R pero que no sea decidible} \Rightarrow \text{Su complementario no es E.R.}}$

Ej: * lenguaje K → $\bar{K}$ NO es E.R   si lo fuera, tendríamos $\left\langle \begin{matrix} \bar{K} \text{ e.r} \\ K \text{ ER} \end{matrix} \right\}$ K decidible => contradicción!
↑ máq que cuando con su propia codif entrada entra en bucle

! Lenguaje decidible : Existe una máq de parada segura que lo reconoce. => E R

! E.R : Existe una máq. que reconoce un lenguaje.

11 Març 05

* Ejercicio 4.4:

L lenguaje

$$\chi_L(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \Rightarrow x \in \overline{L} \end{cases}$$

Función característica:
Aquí no hay máq

$$\chi'_L{}^{(x)} = \begin{cases} 1 & x \in L \\ \uparrow & x \notin L \end{cases}$$

↑ no definida < entra en bucle / para en un estado NO final

Demostrar:

1. L es E.R, sí solo sí, χ'_L es computable (calculable)
2. L es decidible, si solo sí, χ_L es computable.

①. Existe M_L tal que $\mathcal{L}(M_L) = L$ por definición de E.R

$\Rightarrow$)

```
M χ'_L :
    Entrada x
    simular M_L con entrada L
    sí acepta -> escribir 1
    sino Bucle.
```

$\Leftarrow$) $\exists\ M_{\chi'_L}$

```
    Entrada x
    simular M_χ'_L con entrada L
    Aceptar -> (solamente estamos aquí
               si para, si entra en bucle
               nos quedaremos en la línea
               anterior)
```

14 Març 05

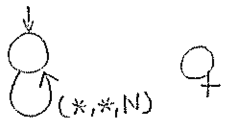
*EJERCICIO 4.1:

¿Es computable la función totalmente indefinida?

$f(x)\uparrow \quad \forall x \in \Sigma^*$

En caso de ser calculable, las entradas para las cuales es indefinida entrará en bucle para cualquier entrada

```
Máquina:    Entrada x
            Bucle ------> Es indefinida porque nunca
                          hay nada en la cinta cuando
                          para xq. nunca para.
```



← máquina que computa la funció buida

$\Rightarrow \forall$ entrada $x \in \Sigma^* \Rightarrow M(x)\uparrow$

*Ejercicio 4.2:

Demostrar que:

$$f(x,y) = \begin{cases} Mx(x)+y & \text{sí } M_x(x)\Downarrow \\ x+y & \text{sino} \end{cases}$$

no es calculable

col dir que si hem de simular → No és una bona solució.

⇓

intuitivament, en el cas $Mx(x)\uparrow$ no podem saber-ho a priori sense simular-la. ⇒ per tant no podem saber que la maq no para.

```
Entrada <x,y>
z := simular Mx(X)      <- si la máq entra en bucle no executarem mai l'alternativa.
sí z es config_terminal -> escribir M(X)+y
sino x+y
```

↑ que esta máq no funcione NO ⇒ que la función no sea calculable ⇒ simplemente no hemos sido capaces de construir una máq para la función (No la hemos encontrado)

* Supongamos que f calculable:

- construir una máq. de parada segura para K.

Mk
Es de parada segura

```
Entrada x
Calcular  z=f(x,0)  (Mf)
si z=x → escribir 0 (Rechazar)   // K̄
sino → escribir 1                // K   z ≠ x
       (Aceptar)
       (Mx(x)↓ en q f)
```

↑
NO acepta K xq podría passar que $Mx(X)=X$

estamos suponiendo que existe una cierta Mf que calcula f definida para todas las entradas ⇒ esa máq para cualquier entrada siempre debe devolver algo
⇒ M_f es de parada segura
(xq sino habría una entrada para la cual NO estaría definida)

$$Mx(X)\begin{cases}(1)\ Mx(X)\uparrow & \Rightarrow \text{Por definición } f(x,y)=x+y\\ (2)\ Mx(X)\downarrow\ \wedge\ Mx(X)\neq X & \rightarrow f(x,y)\neq x+y\\ (3)\ Mx(X)\downarrow\ \wedge\ Mx(X)=X & \Rightarrow f(x,y)=x+y\end{cases}$$

La Máq reconoce:
⇒ Rechazamos cuando $x\in\overline{K}\cup\{x\mid Mx(x)=x\}$
⇒ Aceptamos cuando $x\in K-\{x\mid Mx(x)\neq x\}$
⇒ Esta máq NO reconoce K!

Ara tenemos otro problema ⇒ Demostrar que este lenguaje no es decidible ⇒ + complicado!

---

* Partimos de $G(x)=f(x,1)$, suponiendo que f es calculable

$$G(x)=\begin{cases}Mx(x)+1 & \text{si } Mx(x)\downarrow\\ x+1 & \text{sino}\end{cases}$$

Como f es calculable ⇒ que G es calculable

```
MG { Entrada x
     simular z=f(x,1)
     escribir z
```

Sea x_0 la codificación de G ⇒ $\varphi_{x_0}=G$

$$M_{x_0}(x_0)\begin{cases}M_{x_0}(x_0)\downarrow \Rightarrow \overset{\text{apliquem } G}{M_{x_0}(x_0)+1} \text{ Contradicción!!} \Rightarrow \text{(Tiene otro valor)}\\ M_{x_0}(x_0)\uparrow \Rightarrow M_{x_0}(x_0)=x+1 \text{ Contradicción!!}\end{cases}$$

$M_{x_0}(x_0)$ cuanto vale M_{x_0} evaluado en x_0

* Ejercicio 4.8:

L es E.R, si y solo si, existe una TM M de parada segura tal que

$L=\{M(1), M(0), M(00), \ldots, M(0^n), \ldots\}$ ⇒ L es ∞
(escriu la conf. resultant de simular $M(0^n)$)

⇒) Sea M_L una TM que reconoce L ('No tiene xq ser de parada segura')
Esperamos hasta aceptar n i devolvemos la n-èssima la palabra.
⇒ Construimos M:

```
Entrada 0^(n)   — solo nos interesa saber cuantos
t:=1;
salir = falso
r:=0
mentre no salir
    para ω desde λ hasta t        contamos cuantas palabras acepto i a la n-ésima paro
        simular t pasos de M_L con entrada ω
        si conf. aceptadora → r++
                              si r==n → salir=cierto;
                                        z=ω
                              fsi
        fsi
    t++; fmientras; escribir z
```

⇐) Dada una Entrada X
Buscar X en $M(0^i)$ variando la i.
si se encuentra ⇒ Aceptar

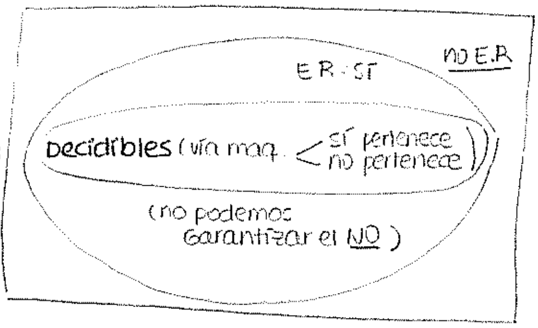
17 Marzo 05

Tema 5: Reducciones

- Problemas $\begin{cases}\text{Decisionales}\\ \text{Funcionales}\end{cases}$

- TM
- Composicionalidad
  Codificación → Church-Turing
  Simulación.
- Lenguajes no decidibles
  Funciones no computables.

- Tenemos tres clases de lenguajes:



* Partiendo de un ejemplo, demostrado en el tema anterior.

K no es decidible
⇓
Halt no es decidible.

⇒) si Halt fuera decidible → M_{HALT} parada segura

$$M_K \begin{cases}\text{Entrada } x\\ \text{si } M_{Halt}(x,x) \text{ acepta} \to \text{Aceptar}\\ \text{sino} \Rightarrow \text{Rechazar}\end{cases}$$

M_{Halt} no existe porque no existe M_K

Suponemos la transformación: $\underset{K}{\langle x\rangle} \to \underset{HALT}{\langle x,x\rangle}$

* ⇒ FUNCIÓN de reducción:

$\Rightarrow \boxed{x\in K \Leftrightarrow \langle x,x\rangle \in HALT} \Rightarrow r\langle x\rangle = \langle x,x\rangle$

⇒) $x\in K \to M_x(x)\downarrow \Rightarrow \langle x,x\rangle \in HALT$

⇐) $x\in K \leftarrow M_x(x)\downarrow \Leftarrow \langle x,x\rangle \in HALT$

Demostramos que son prácticamente equivalentes independientemente de si existe una máq de parada segura que reconozca los lenguajes.

- Diremos que $A\subseteq\Sigma^*$ es reductible (se reduce a) $B\subseteq\Sigma^*$ cuando existe una función $r: \Sigma^* \to \Sigma^*$ tal que:

  1. r es calculable y total (está definida por todas las entradas)
  2. Transforma entradas (palabras) de A en entradas de B.

  ↓ $\forall w\in\Sigma^* \quad w\in A \Leftrightarrow r(w)\in B$

  r es la función de reducción

  $\boxed{\text{Notación: } A \leq_m B}$

EX: A = PARES B = {0}

$$r(x) = \begin{cases} 0 & \text{si } x \text{ es par}\\ x & \text{sino}\end{cases}$$

PARES $\leq_m$ {0}

$\Rightarrow$ Demostrar que HALT no es decidible por REDUCCIÓN:

M_K
```
Entrada x
y := r(X);
sí M_Halt(y) acepta
sino rechazar
```

Es una TM
si M_{Halt} es de parada segura M_K tb
si $x \in K \rightarrow$ Acepta

$\Downarrow$ Tenemos una M_K. contradicción! sabemos que M_K no existe! $\Rightarrow$ M_{Halt} no existe

- Otra forma: usando 2 lenguajes cualesquiera.

$r: \Sigma^* \rightarrow \Sigma^*$

$A \leq_m B$ $\quad x \in A \Leftrightarrow r(x) \in B \Leftrightarrow x \in A.$

$\overset{\Downarrow}{K}$ $\quad \overset{\Downarrow}{HALT}$

A no es decidible $\Rightarrow$ B no es decidible $\rightarrow$ si B lo fuera existiría M_B.

M_A:
```
Entrada x
y := r(X);
sí M_B(y) → Aceptar
sino rechazar
```

$\rightarrow$ M_A acepta cuando $y \in B \Leftrightarrow r(x) \in B$

$\Downarrow$ $y \notin B \Leftrightarrow r(X) \notin B \Leftrightarrow x \notin A.$

$\boxed{\text{Teorema: sea } A,B \subseteq \Sigma^* \text{ tal que } A \leq_m B \text{ Entonces: } A \text{ no decidible} \Rightarrow B \text{ no decidible.}}$

$\boxed{A \text{ no E.R} \Rightarrow B \text{ no E.R}}$

Ejemplo: A no es E.R $\Rightarrow$ B no es E.R. si B fuera E.R existiría M_B ↗

$\left\{ \begin{array}{ll} B \text{ decidible} & \Rightarrow A \text{ decidible} \\ B \text{ ER} & \Rightarrow A \text{ ER} \end{array} \right\}$ $\quad A \leq_m B \Rightarrow B$ es al menos tan difícil como A !

Reducción $\Rightarrow$ Relación de dificultad !

$\rightarrow$ Esquema demostración HALT no es decidible.

$\left. \begin{array}{l} \boxed{K \leq_m HALT} \text{ con } r(\langle x \rangle) = \langle x, x \rangle \\ K \text{ es no decidible} \end{array} \right\}$ $\Rightarrow$ HALT no Decidible.

Ahora demostrar que $r(\langle x \rangle)$ es función de reducción.

Ejemplo 2:

HelloWorld
HW = {x | Mx escribe "Hello world" con cualquier entrada}
K no decidible.
$\boxed{K \leq_m HW \quad ¿r(x)?}$ si encontramos la reducción habremos demostrado que HW es no decidible.

r calculable y total
$x \in K \Leftrightarrow r(x) \in HW$

Buscamos una función que transforme máquinas.
K y HW son un conjunto de máquinas.

```
Entrada <x,w>
simular Mx(x)
Aceptar
```

sí $x \in K$, $M_x(x)\downarrow$ y acepta la entrada
sí $x \notin K$, $M_x(x)\uparrow$ (Entra en bucle)

$\Rightarrow$ Hemos encontrado una máquina que solamente para sí $x \in K$. $\Rightarrow$ Estamos reduciendo el problema

Entrada $\langle x, \omega \rangle$
Simular $Mx(X)$
Escríbir "Hello world"

} Esta máquina tiene un doble comportamiento $\rightarrow$ Es lo que buscamos

$\downarrow$
Aquí la x es la entrada de la máq. y en r(x) no $\Rightarrow$ Para convertir la máq en función haremos:

$r(X) =$ Entrada $\langle \omega \rangle$
Simular $Mx(X)$
Escríbir "Hello World"

↙ Para cada $\langle x \rangle$ tenemos una máq.

HW es un conjunto de codificaciones de máquinas. Falta codificar la máquina para que sea HW.

$$r(X) = \text{cod}\begin{pmatrix}\text{Entrada } \langle \omega \rangle \\ \text{Simular } Mx(X) \\ \text{Escríbir "Hello world"}\end{pmatrix}$$

$\uparrow$ $Mr(X)$

Treure la x a fora $\rightarrow$ Teorema de la parametrització.

Debemos trabajar en codificaciones!

Hay que demostrar que r es calculable y total
- Total: Porque dada una codificación de máq. siempre existe $Mr(x)$.
- Calculable: Dada una codificación puedo devolver ese valor, construír la TM y codificar-la.

Teorema: s-m-n (Teorema de la parametrización)

- $\overset{\text{Hipótesis}}{x \leq K} \Rightarrow r(X) \in HW \underset{\text{def.}}{\Leftrightarrow} Mr(X)$ escribe "Hello World".

sí $x \in K$, la máq. acepta i escríbe "Hello world". $\rightarrow$ demostrado!
si $x \notin K$, $\Rightarrow r(X) \notin HW \Leftrightarrow Mr(X)$ no escribe "Hello world"
$\uparrow$ entra en bucle.

Hemos demostrado que r(x) es función de reducción!

Ejemplo 3:

↙ Como mínimo hay una palabra que acepta.

$DOM = \{\langle x \rangle \mid \mathcal{L}(Mx) \neq 0\} = \{\langle x \rangle \mid \exists \omega \; Mx(\omega)\downarrow \text{ y acepta}\}$

Demostración que no es decidible por reducciones a un lenguaje que no lo és.

$K \leq_m DOM$ Transformar máq. que para con su própia configuración en máq. que para en alguna.

sea la codificación
$$\begin{pmatrix}\text{Entrada} \langle z \rangle \\ \text{Simular } Mx(x) \\ \text{Aceptar}\end{pmatrix} = r(X)$$

r(x): Es calculable $\longrightarrow$ El cálculo de las codificaciones es un algoritmo lo garantiza el teorema s-m-n
Es total $\longrightarrow$ toda x tiene máq. y toda máq. tiene codif. por definición.
$x \in K \Leftrightarrow r(X) \in DOM \longrightarrow$
$x \in K \Leftrightarrow Mx(X)\downarrow \Rightarrow \mathcal{L}(M_r(x) = \Sigma^* \Rightarrow$
$\mathcal{L}(Mr(x)) \neq 0 \Rightarrow r(x) \in DOM$

$x \notin K \Rightarrow Mx(x)\uparrow \Rightarrow \mathcal{L}(Mr(x)) = 0 \Rightarrow r(X) \notin DOM$.

18 Març 05.

$\boxed{\text{Exercici 4.15.}}$ consideremos el problema:

Dada una TM M y una entrada ω, decidir si M con entrada ω se para immediatamente después de escribir 'Aquí m'aturo'
¿Decidible? ¿E.R?

* Intuitivament: tenim un llenguatge parella màq / entrada

si formalitzem el llenguatge: $L = \{ <x,\omega> \mid M_x(\omega)\downarrow$ immediatament després d'escriure 'aquí m'aturo' $\}$

⇒ si sembla ER – anem simulant passos i mirem a la cinta, si el que hi ha a la cinta en un cert pas i és la frase 'aquí m'aturo' només caldrà analitzar el següent pas ⇒ x veure si para o no $M_x(\omega)$. ⇒ Trobem una màq

⇒ No és decidible – anem esperant que escrigui la frase 'aquí m'aturo' i aquesta no s'escrigui mai ⇒ entrarem en bucle

⟹ $\boxed{\text{Demostrem } \underline{\text{NO}} \text{ decidible.}}$ Busquem una reducció desde K al nostre llenguatge

⇒ Hem de trobar $r(x)$ – calculable
– total
– $\underset{\text{elements de } K}{x \in K} \Leftrightarrow \underset{\text{elements de } L}{r(x) \in L}$

$r(x): \Sigma^* \rightarrow \underset{\text{parelles codif màq-paraula}}{\Sigma^*}$

imatge

$$r(x) = \Big(\underbrace{\text{cod} \begin{pmatrix} \text{entrada } \omega \\ \text{simular } M_x(x) \\ \text{Escriure 'aquí m'aturo'} \end{pmatrix}}_{y(x)} , \underbrace{\lambda}_{\omega} \Big)$$

$M_{y(x)}$

↑ extreure la codif. d'una màq coneguda ⇒ <u>calculable</u> Teorema s-m-n

<u>total</u> perquè per cada entrada de x tenim una imatge.

$x \in K \Leftrightarrow r(x) \in L$

$x \in K \Rightarrow \underset{\text{immediatament després d'escriure 'aquí m'aturo'}}{M_{y(x)}\downarrow} \Longrightarrow \underset{\text{immediatament després d'escriure 'aquí m'aturo'}}{M_{y(x)}(\lambda)\downarrow} \Leftrightarrow r(x) \in L$

$x \notin K \Rightarrow M_{y(x)}(\omega)\uparrow \Rightarrow M_{y(x)}(\lambda)\uparrow \Leftrightarrow r(x) \notin L$

$\boxed{\text{* Exercici 4.13:}}$ consideramos el problema:

– Condicional:
si para – l'element és de L
si no para – mirar compliment condició

Dada una TM, decidir si nunca $\overset{\text{devuelve}}{\text{escribe}}$ (cuando para) una palabra de longitud superior a la de la entrada.
¿ES E.R? ¿ES DECIDIBLE?

Està format únicament x màquines, ja que s'ha de verificar x totes les entrades, no per cap en concret

Formalitzem el llenguatge ⇒ $L = \{ <x> \mid \forall \omega \in \Sigma^* \; M_x(\omega)\downarrow \wedge |M_x(\omega)| \leq |\omega| \vee M_x(\omega)\uparrow \}$

$\overline{L} = \{ <x> \mid \exists \omega \; M_x(\omega)\downarrow \wedge |M_x(\omega)| > |\omega| \}$

Intuitivament: No és E.R No es podrà detectar que $M_x(\omega)\uparrow$ (no para) com tenim Σ^* (infinit) no es podrà comprovar per totes les entrades. ⇒ No acabarà reconeixent L

En canvi, $\overline{L}$ és E.R ⇒ Existeix màquina que localitza una paraula que compleix la cond.

$\left.\begin{matrix} L \text{ ER} \\ \overline{L} \text{ E.R} \end{matrix}\right\}$ ⇒ L decidible. si demostrem que L no es decidible L no és E.R perquè $\overline{L}$ ho és

⇒ $K \leq_m \overline{L}_{②} \Leftrightarrow \underset{①}{\overline{K}} \leq_m L$ (sabent que $\overline{K}$ no es E.R)] són la mateixa reducció.

18 Març 05

Nota:

! $\boxed{A \leq_m B \iff \bar{A} \leq_m \bar{B}}$

$M_{r(x)} = M_y(x)$

$r(x) = \text{cod} \left(\begin{array}{l} \text{Entrada } \omega \\ \text{simula } M_x(x) \\ \text{escribir } \omega \cdot a \end{array} \right)$ → escribe algo + gran

- $x \in \bar{K} \iff r(x) \in L$

  $\forall x\ ) \Rightarrow r(x)$

  $x \in \bar{K}_y \Rightarrow M_x(x)\uparrow \Rightarrow \underset{\forall \omega}{M_{r(x)}(\omega)\uparrow} \Rightarrow r(x) \in \bar{L}$ ja que $\forall \omega\ M_y(\omega)\uparrow$ condició del L

  $x \in K_y \Rightarrow M_x(x)\downarrow \Rightarrow M_{r(x)}(\omega) \underset{\forall \omega}{=} \omega \cdot a \Rightarrow r(x) \notin L \equiv r(x) \in \bar{L}$

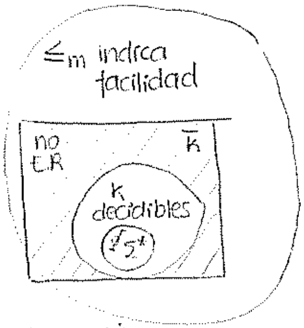
---

31 Març 05.

f función de reducción $A \leq_m B$

- f calculable
- f total
- $\forall \omega \quad \omega \in A \Leftrightarrow f(\omega) \in B$

$A \leq_m B$, si A no decidible ⇒ B no decidible
A no E.R. ⇒ B no E.R



los lenguajes decidibles son los más fáciles en términos de computabilidad

Proposición: A decidible, $B \neq \emptyset$, $B \neq \Sigma^*$ cualquiera $\Big\} \Rightarrow A \leq_m B$

Demostración:

$B \neq \emptyset \Rightarrow \exists b_0 \in \Sigma^*,\ b_0 \in B$

$B \neq \Sigma^* \Rightarrow \exists \bar{b}_0 \in \Sigma^*,\ \bar{b}_0 \notin B$

sea $f(\omega) = \begin{cases} b_0 & \text{si } \omega \in A \\ \bar{b}_0 & \text{si } \omega \notin A \end{cases}$

A decidible ⇒ $\exists M_A$ de parada segura $\mathcal{L}(M_A) = A$

f es total

$\omega \in A \Rightarrow f(\omega) \in B$
$\omega \notin A \Rightarrow f(\omega) \notin B$

f es calculable

$M_f : \left(\begin{array}{l} \text{Entrada } \omega \\ \text{simular } M_A \text{ con entrada } \omega \\ \text{si Acepta} \rightarrow \text{escribir } b_0 \\ \text{sino escribir } \bar{b}_0 \end{array} \right)$

Proposición: $A \leq_m B \Leftrightarrow \bar{A} \leq_m \bar{B}$

⇒ : $\exists f$ calculable y total tq $\forall \omega \in \Sigma^*\ \omega \in A \Leftrightarrow f(\omega) \in B$

contrarecíproco

$\left. \begin{array}{l} \omega \notin A \Rightarrow f(\omega) \notin B \\ \omega \notin A \Leftarrow f(\omega) \notin B \end{array} \right\}$ $\boxed{\omega \notin A \Leftrightarrow f(\omega) \notin B}$ $\equiv$ $\boxed{\omega \in \bar{A} \Leftrightarrow f(\omega) \in \bar{B}}$

Demostrado!

⇐ : $\bar{\bar{A}} = A$

Nota: $K \leq_m A \Rightarrow$ A no es decidible
$\bar{A}$ no es E.R

— Teorema de Rice :

- — Nueva herramienta para algunos problemas no decidibles
- — Idea del porque existen cosas no decidibles → cuando necessita averiguar cosas de la semàntica de la màquina → las cosas no funcionan → NO decidibles
- Trabajan sobre conjuntos de índices
- Hay ∞ máq. que hacen la misma tarea en el fondo

- A es un conjunto de índices sí $\forall x,y \in \mathbb{Z}^*$ se cumple la siguiente propiedad:

$$x \in A \ \wedge \ \varphi_x = \varphi_y \ \Rightarrow \ y \in A$$

- Es un conjunto de codificaciones de máquinas
- Las máquinas que calculan la misma función, o estan todas en A o no está ninguna.

* Ejemplos: $DOM = \{ x \mid \mathcal{L}(M_x) \neq \emptyset \}$

No es conjunto de índices → $\overset{\text{NO}}{\text{Encontrar}}$ x,y $x \in A \wedge \varphi_x = \varphi_y \wedge y \notin A$

M_x:

```
Entrada ω
sí ω es par
    escribir 1
    Aceptar
sino
    escribir 0
    Rechazar
  ↓
Reconoce el
lenguaje de
Pares
```

M_y:

```
Entrada ω
sí ω es par
    escribir 1
    Rechazar
sino
    escribir 0
    Rechazar
  ↓
Reconoce ∅
```

$\varphi_x = \varphi_y$, pero $\mathcal{L}(M_x) = \text{PARES}$ y $\mathcal{L}(M_y) = \emptyset$

$x \in DOM$ $\qquad$ $x \notin DOM$

$FDOM = \{ x \mid DOM(\varphi_x) \neq \emptyset \}$ sí es conjunto de índices

1) $A = \{ x \mid \varphi_x$ es total $\}$ es un conjunto de índices
2) $B = \{ x \mid \varphi_x$ es de parada segura $\}$ es un conjunto de índices

1) Sea x,y tal que $\left.\begin{matrix} x \in A \\ \varphi_x = \varphi_y \end{matrix}\right\} \overset{?}{\Rightarrow} y \in A$

φ_x es total $\Rightarrow$ φ_y es total $\Rightarrow$ $y \in A$ Demostrat!

31 Març 05

Teorema de Rice : sea A un conjunto de índices. Entonces :

A es decidible $\Leftrightarrow A = \emptyset \vee A = \Sigma^*$

Los únicos conjuntos de índices decidibles son: $\emptyset$ y Σ^*.

! si A es conjunto de índices $\Rightarrow \bar{A}$ también lo es.

Demostración del T. Rice:

$\Leftarrow$) Trivial

$\Rightarrow$) : A es conjunto de índices $\Rightarrow \forall x,y \in \Sigma^* \; x \in A \wedge \varphi_x = \varphi_y \rightarrow y \in A$

Supongamos que $A \neq \emptyset \wedge A \neq \Sigma^* \overset{?}{\Rightarrow} A$ es no-decidible.

$\exists a \in A$

$\exists \bar{a} \notin A$

Para demostrar que A es no-decidible $\Rightarrow$ Reducciones :

$K \leq_m A$

a)

$$f(x) = cod \underbrace{\begin{pmatrix} \text{Entrada } \omega \\ \text{simular } M_x(x) \\ \\ \text{simular } M_a(\omega) \\ \text{Escribir } M_a(\omega) \end{pmatrix}}_{M_{f(x)}}$$

si $x \in K \Rightarrow \varphi_{f(x)} = \varphi_a$ (Hace lo mismo que A)
$a \in A$
misma función calculada
$\Rightarrow f(x) \in A$ xq. A es un conjunto de índices

si $x \notin K \Rightarrow$ Problema! $\varphi_{f(x)} \uparrow \; \forall \omega$ debido a que $M_{f(x)}$ no para.

$\overset{?}{\Rightarrow} f(x) \notin A$

Hasta donde queremos llegar para una reducción valida.

a) $c \notin A$

sea c una codificación de una máquina que nunca para

$\Rightarrow M_c(\omega) \uparrow \; \forall \omega$ $\leftarrow$ reconoce función vacía
$\varphi_c(\omega) \uparrow$

$c \in A$: $x \notin K \Rightarrow \varphi_{f(x)} \uparrow \; \forall \omega) \Rightarrow f(x) \notin A$

$\equiv$ $\quad$ $c \notin A$, y A es conj de índices

$\varphi_c \uparrow \Rightarrow \boxed{\varphi_{f(x)} = \varphi_c}$

b) $c \in A$

$$f(x) = cod \begin{pmatrix} \text{Entrada } \omega \\ \text{Simular } M_x(x) \\ \\ \text{simular } M_{\bar{a}}(\omega) \\ \text{Escribir } M_a(\omega) \end{pmatrix}$$

$x \in \bar{K} \Rightarrow \; \dots \Rightarrow f(x) \in A$

$x \notin \bar{K} \Rightarrow \; \dots\dots \Rightarrow \boxed{f(x) \notin A} \Rightarrow \varphi_{f(x)} = \varphi_{\bar{a}}$

$\boxed{\bar{K} \leq_m A}$

$\Uparrow$

Reducción.

$\bar{a} \notin A$

$\Rightarrow f(x) \notin A$

Corolario: si A es un conjunto de índices que contiene las codificaciones de las maquinas que calculan la función vacía ($f(\omega) \uparrow \; \forall \omega$) entonces A no es E.R

1 Abril 05.

Tema 6. Algunos problemas indecidibles 'clásicos':

El problema de reescritura (palabras de Thue) - WP:

"Dada una lista finita R de reglas de reescritura (substitución) $R = \{ (u_j, v_i) \mid u_j, v_i \in \Sigma^* \}$ y dos palabras $u, v \in \Sigma^*$, determinar si es possible construir v a partir de u usando R."

Ejemplos:

* Reglas:

$\overset{u_i}{(a,} \overset{v_i}{aa)}$
(bb, b)
(aaaaaa, bbbb)

u = aba
v = ba

aba → aaba → aaaba → aaaab^a → aaaaaba → aaaaaaba → bbbbba → bbbba → bbba → bba → ba

* Reglas:

(a, aa)
(bb, b)
(aaaaaa, bbbb)

u = bbb
v = a

} No tiene solución ⇒ No tenemos ninguna regla que transforme b's en a's.

- Teorema: El problema de reescritura es no-decidible

Demostración:

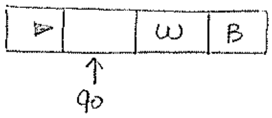
Construiremos una reducción desde un lenguaje no-decidible

PERT = $\{ (x, w) \mid w \in \mathcal{L}(M_x) \}$

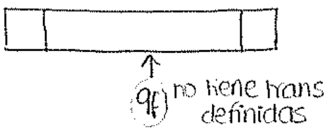
Demostrar que: $P \leq_m WP$

f calculable
f total
$z \in PERT \Leftrightarrow f(z) \in WP$

- configuración inicial de la máquina



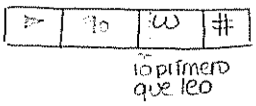
- configuración donde aparece q_f



no determinista.

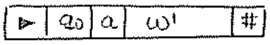
Decidimos que representamos las configuraciones:

1 colocamos el estado y la palabra que está a su derecha es la que hay en la cinta.
2. Para acabar usamos un símbolo especial.

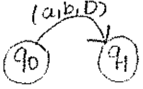


→ Esta es nuestra palabra inicial

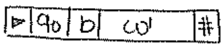
Si todo va bien al final tendremos:



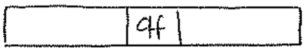
si la máquina es $w = aw'$



→ Aplicamos transición:



Esto lo podemos hacer para todas las transiciones de la máq y llegaremos a:



Hacemos reescritura de palabras ⇒ tenemos una cadena de substituciones ⇒ si escribimos una regla de reescritura para cada una de las transiciones y consideramos la palab inicial. si aplicamos todas las reglas sobre ω_0 y obtenemos $\omega f = \sigma$ cumpliremos el problema de la reescritura.

⇒ Demostración formal:

Sea la función de reducción definida como:

Reglas reescritura:

$\forall p,q \in Q$ y $\forall a,b,c \in \Gamma$

si $\delta(q,a) = (p,b,D) \rightarrow (qa, bp)$ ①

si $\delta(q,a) = (p,b\ N) \rightarrow (qa, pb)$

si $\delta(q,a) = (p,b,I) \rightarrow (cqa, pcb)$ ②

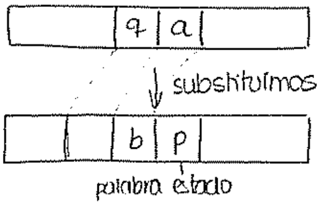
si $\delta(q,B) = (p,b,D) \rightarrow (q\#, bp\#)$

si $\delta(q,B) = (p,b,N) \rightarrow (q\#, bp\#)$

si $\delta(q,B) = (p,b,I) \rightarrow (cq\#, pcb\#$

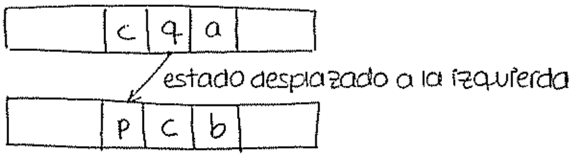
$\boxed{u = \triangleright q_0 \omega \#}$

①. si en la cinta nos encontramos



Hemos ido a la derecha, xq. lo que vemos esta a la derecha de p y hemos desplazado el estado a la derecha

②.



Estamos simulando el comportamiento de la máquina ⇒ PERT

- si llegamos a qf estamos en un lugar donde no sabemos que hay delante y detrás de qf en la cinta : $\overset{?}{\ }\boxed{q_f}\overset{?}{\ }$

Ahora, eliminaremos lo que hay ≠ qf con las reglas de reescritura

$$\Rightarrow_{(*)}\left\{\begin{array}{l}(aq_f, q_f)\\(q_fa, q_f)\end{array}\right\}$$

⇒ Entonces $\sigma = q_f$. si ω inicial pertenece al final de la máquina quedará $\boxed{\ \ q_f\ \ }$ y luego aplicando (*) llegaremos a σ. Como habremos conseguido u ⇒ σ la máquina habrá reconocido la palabra de entrada → PERT

$$\underbrace{f(x,\omega)}_{\text{entrada de } M_{PERT}} = \underbrace{(R,u,v)}_{\text{entrada de } M_{WP}}$$

4 Abril 05

Altre professor: Guillem Godoy
Despatx: Omega 113.
Consultes: Dijous 12-14h

- Problema 4:

PCP:

- Entrada: llista parells paraules: $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle \dots \langle u_n, v_n \rangle$ (Tenim 2 camps d'entrada)
- Sortida: Podem fer una elecció de parells tals que concatenem les parts esquerres i les dretes respectivament

Exemple: $\langle aab, a \rangle$
$\langle ba, abc \rangle$
$\langle baa, ba \rangle$

$u_1 u_2 u_1 \rightarrow aab\ ba\ aab$
$v_1 v_2 v_1 \rightarrow a\ abc\ a$

⇒ Donar resposta afirmativa si puc fer una elecció de parells, tals que, concatenant parts esquerres i dretes successives obtenim la mateixa paraula (en el mateix ordre)

Demostrar que PCP és indecidible ⇒ No existeix cap programa que permet solucionar-lo.

Quan la longitud de la part esquerra és 1 ⇒ llavors ≡ "Gramàtiques"

WP: Problema reescriptura de mots
Entrada: llista de regles de reescriptura de paraules i dues paraules addicionals, u, v.
Resposta: És pot arribar de u a v aplicant reescriptura.

agafem una entrada qualsevol de WP i la passem com a entrada de PCP preservant la resposta.
⇒ Sabem a priori que WP és indecidible
→ PCP donada una entrada WP dóna una sortida PCP (llista)
⇒ suposem que PCP és decidible ⇒ podem arribar a trobar un programa que soluciona WP ⇒ Hem preservat les respostes i ⇐⇒ estem solucionant un problema WP tq que ja sabiem que no era decidible ⇒ contradicció.

$u_1 \rightarrow v_1$
$u_2 \rightarrow v_2$
Podem arribar de u_1 a v_1 fent reemplaçaments? ⇒ El meu sistema de parelles fem que simuli aquest comportament

⇒ 1ª Idea ⇒ En principi elecció $\langle u, \lambda \rangle$ ⇒ Tenim una llista de paraules que són reescripcions de u ⇒ Necessitem una sèrie de separadors (fer ex #) → u#...

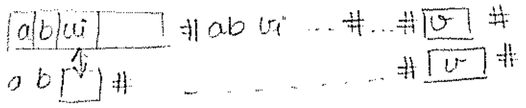
→ Per tots els símbols del alfabet (a,a) (b,b) (c,c). Ens interessa en un moment donat poder fer una reescriptura

$\overbrace{a\,b\,u_i \qquad}^{u}$ # $a b v_i$

... després de moltes reescriptures ⇒ Apareixerà v ⇒ un cop aconseguit v# per acabar

Regles que simulen reescriptura
$(u_1, v_1) \longrightarrow (u_m, v_m)$ ∀ símbol
⋮
$(\#, v\#) \longrightarrow (\#, \#)$

⇓



Inventem nous símbols (a, a') que gràficament són diferents però semànticament iguals (espècie de còpies)

Estat Global Màq → Configuració. – Estat Autòmat + Contingut de la cinta + Posició del capçal

Donada qualsevol TM, modificant les transicions que salten a estat final (i per les no definides) per transicions a un estat que esborra tota la cinta i després saltem a un estat final únic, aconseguim que totes les configuracions d'acabament siguin la mateixa.

Estat Global Màquina – CONFIGURACIÓ → Estat Autòmat + Contingut de la cinta + Posició del capçal.

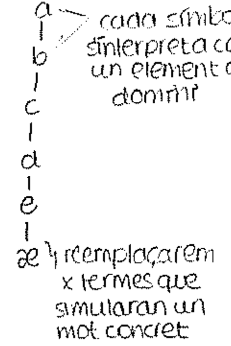
TM determinista – la pròpia màquina no pot escollir el camí d'execució i només pot accedir des de l'origen a 1 estat concret per un camí propi del fil d'execució.

- **Problema 5.** P(x, terme(x,y))

═ Terme – Arbre sintàctic (amb símbols als nodes)

terme (a b c d e, x) ← variable

↓ Tenim un arbre amb símbols unaris. Tenim una nova constant c' que l'afegim en lloc de la variable

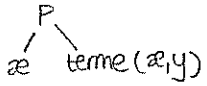


cada símbol s'interpreta com un element del domini

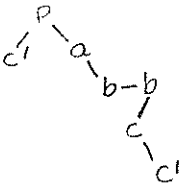
reemplaçarem x termes que simularan un mot concret

═ P(x, terme(x,y)):

En la lògica tenim símbols de Predicat P, Q .... amb una certa aritat → si P és binari el podem representar com un arbre (P₂)



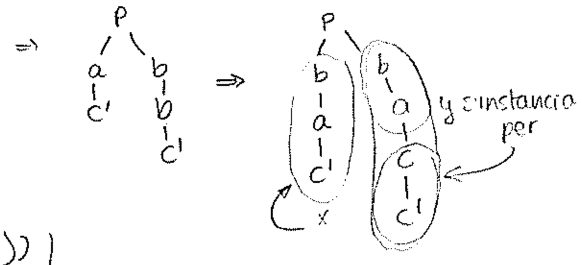
– Semànticament:
Aquest predicat es una propietat que s'avalua a cert o a fals   P(x, terme(abc, x))



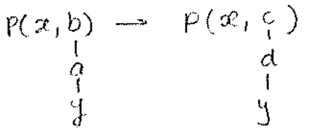
Regla reescriptura
ba → cd

⇒ Podem fer reescriptures que permeten anar movent símbols d'una branca a l'altra de P

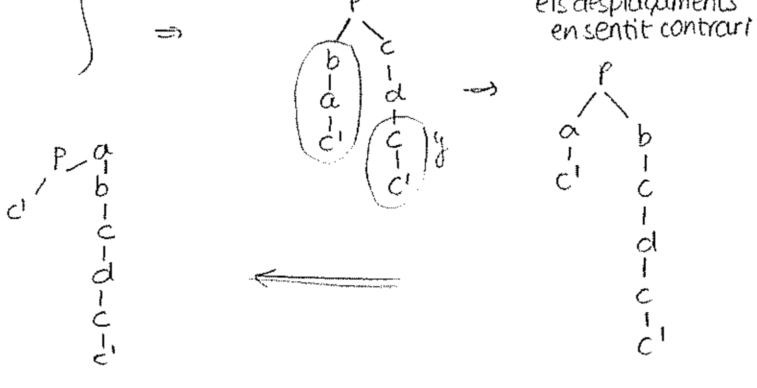
Clausula:
P(x, terme(a,y)) →
P(terme(a,x), y)



P(x, terme(ui,y)) → P(x, terme(vi,y))



instanciació



4Abril05.

PART 2

Tema 8 – Gramàtiques

! Reescriptura de Mots '≡' Màq. de Turing.

Gramàtica ≡ Màq. indeterminista amb Estats i pila. → limitació de les TM u, v

Gramàtica: Reescriptura de paraules $u \to v$ quan $|u| \to 1$. Cas particular de "reescriptura de mots" de parada segura.

A vegades, amb una gramàtica es vol fixar un símbol com a punt de partida (símbol inicial), i llavors es parla dels mots accessibles des d'aquest com el llenguatge generat per la gramàtica. (*)

*Exemple de gramàtica:

fixem com a símbol inicial

$$\begin{cases} E \to E+E \mid E*E \mid N \\ N \to DN \mid D \\ D \to 0 \mid 1 \mid 2 \mid \dots \mid 9 \end{cases}$$

→ E Representa 3 regles de reescriptura: $E \to E+E$, $E \to E*E$, $E \to N$

Sistema de reescriptura de mots:

$$\underline{E} \to \underline{E}*E \to E*\underline{E}+E \to E*\underline{N}+E \to E*D\underline{N}+E \to E*D\underline{D}+E \to E*\underline{D}2+E \to E*12+E$$

(*) Però generalment es considerarà com a llenguatge generat ~~més~~ no més als mots accessibles que continguin símbols TERMINALS (símbols que no tenen representació en la part esquerra de les regles).

- estem restringint
- podem representar menys coses (llenguatges)

↓ Reescriptura de Mots '≡' Màq. Turing

↓ Gramàtica ≡ Màq. indeterminista amb Estats i pila → limitació TM

↓ Gramàtiques lineals ≡ màq. que només tenen estats i entrada no modificable ⇒ AUTÒMATS.

- Guanyem calculabilitat
- Guanyem complexitat i altres propietats

una gramàtica és un sistema de reescriptura de mots on les parts esquerres tenen mida 1.

Interpretació llista de instruccions → $\boxed{L \to I;\ (L \mid I)}$ ← símbol inicial

Definim ara un sistema de reescriptura de mots, amb 2 regles:

$L \to \boxed{I;L}$

$L \to \boxed{I;}$

Que s'utilitza per reemplaçament.

$L \to I;\ \underbrace{(L \to I;I)}_{\text{reemplaçament}};\ L \to I;I;I;$

$L \to I;\ Id := E;I;I;L$

$I \to Id\ (":=")\ E$

I ← instrucció; Id: variable; ":=" → únic símbol que significa assignació; E: de l'exemple anterior

llista de símbols que interpretem com a instruccions del llenguatge:

If E then L |

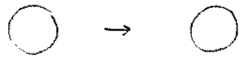
If E then L else L | ← L — Definició d'un llenguatge de programació

while E do L

$L \to I;(L) \to$

$\to I;\ \underline{I;}(L) \to I\ (I)\ I;L \to$

$\to I;\ \underbrace{Id = E}\ ;IIL$



variables (símbols constructors) símbols part esquerra.

símbols terminals (Id, if, while ...)

Els símbols que apareixen a les parts esquerres s'anomenen VARÍABLES (majúscules); i els altres s'anomenen símbols Terminals (minúscules). El llenguatge definit per (o associat a) una Gramàtica és el conjunt de paraules accessibles per a reescriptura des del símbol (variable) inicial.

- Trobeu Gramàtiques pels llenguatges següents:

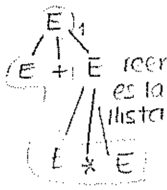
a) $\{a^nb^n \mid n\geq 0\}$
b) $\{a^nb^n \mid n\geq 1\}$
c) $\{a^ib^j \mid i>j\}$
d) $\{a^ib^j \mid i<j\}$
e) $\{a^ib^j \mid i>2j\}$ $\{a^{\alpha}a^jb^j \mid \alpha\geq 1, j\geq 0\}$
f) $\{a^ib^j \mid 2j>i\}$
G) $\{a^ib^j \mid i>j \vee i<2j\}$
h) $\{a^ib^j \mid j<i<2j\}$
i) $\{a^ib^j \mid i\neq j \wedge i\neq 2j\}$

(j) $\{a^ib^jc^k \mid i=j+k\}$
(k) $\{a^ib^jc^k \mid i\geq j+k\}$
(l) $\{a^ib^jc^k \mid 2i=j+k\}$
(m) $\{a^ib^jc^k \mid i=2(j+k)\}$

$S \to S_bB$
$B \to bB \mid b$
$S_b \to aS_bb \mid \lambda$

Propietat de les Gramàtiques:
Ambigüitat: si una mateixa paraula por generar-se de maneres diferents.

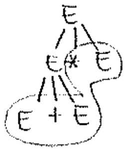
Arbre de derivació:
$E \to E+\underline{E} \to \boxed{E+E*E}$



reemplaçament E₁ es la corresponent llista d'anels directes

En tot moment el recorregut d'esquerra a dreta de les fulles és la paraula final resultant de la derivació.

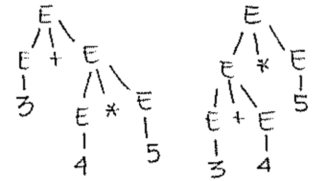
Existeix un arbre de derivació alternatiu per la mateixa paraula: $E \to E*E \to E+E*E$



⇒ existeix + d'un arbre de derivació.

Definició: Diem que una gramàtica és ambigua si existeixen 2 arbres de derivació diferents per a una mateixa paraula

3+(4*5)   (3+4)*5



↳ correspón a la parentització de les paraules.

→ (a) $\{a^nb^n \mid n\geq 0\}$
símbol inicial Gramàtica

$\boxed{S \to aSb \mid \lambda}$

Paraules llenguatge $\{a_1[\overset{\text{començar per a}}{}\dots a_nb_1\dots]b_n\}$ totes tenen exactament la mateixa forma excepte el cas $a^0b^0=\lambda$ paraula buida.

λ
ab
$aabb$
$aaabbb$
$\vdots$

$S \to a\underline{S}b \to aa\underline{S}bb \to \dots$
si és λ Generem ab
si és λ Generem $aabb$
Generem tot el llenguatge

→ (b) $\{a^nb^n \mid n\geq 1\}$
$S \to aSb \mid ab$

→ (c) $\{a^ib^j \mid i>j\}$

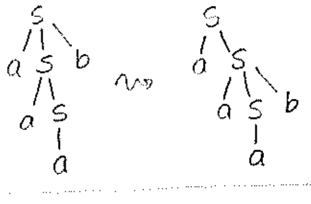
En els casos:
$\{ \_\_ \mid \sim\sim \vee \sim\sim \}$
$S \to S_1 \mid S_2$

c1) $S \to aS \mid aSb \mid a$ Podem concatenar llenguatges
↑ és ambigua

$a\boxed{\overset{\lambda}{\dots ab\dots}}b$

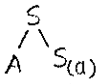
G) $S \to S_1 \mid S_2$
$S_1 \to aS_1b \mid a$
$S_2 \to$

aaab



c2) $S \to \underset{\smile}{A}S_{(a)}$ Gramàtica Apartat A
A: llenguatge que conté A's

$S_{(a)} \to aS_{(a)}b \mid \lambda$
$A \to aA \mid a$

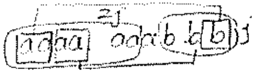


NO ambigu

$\{a^ib^j \mid i>j\} \equiv \{a^{j+\alpha}b^j \mid j\geq 0; \alpha\geq 1\}$
$\equiv \{a^{\alpha}a^jb^j \mid j\geq 0\}$
$\equiv \{a^{\alpha} \mid \alpha\geq 1\}\{a^jb^j \mid j\geq 0\}$
↑ desambigüem

→ (e) $S \to AS'$
- $S' \to aaS'b \mid \lambda$
  $A \to aA \mid a$
- $S \to aaSb \mid A$
  $A \to aA \mid a$

Hi ha una partició on tenim el doble del nombre de a's.
$\underbrace{a\dots a}_{2j}\underbrace{b\dots b}_{j}$



Agafem les primeres a's associant-les amb les b's del final.

1 si las 2 arbres diferents aplicant diferents cops la Gramàtica ⇒

Tindrem arbres dif tals que un tindrà més b que l'altre ⇒ tindrem paraules diferents.

- $S \to aaSb \mid aS \mid a$ → És ambigua

x Raonament NO ambigua:

Els arbres de derivació diferents corresponen al nombre que apliquem la Gramàtica i obtenim paraules diferents ⇒ NO ambigu

8 Abril 05

→ (f) $\{a^ib^j \mid 2j>i\}$

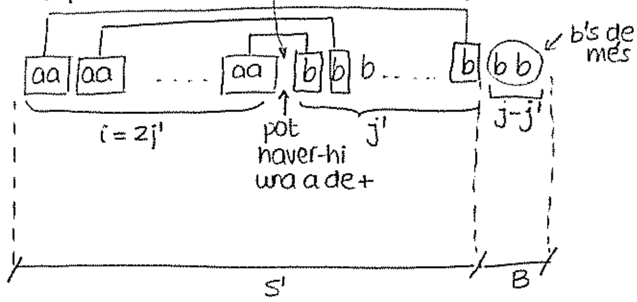
- $S \to S'B \mid aS'B$
  $S' \to aaS'b \mid \lambda \mid a$
  $B \to bB \mid b$

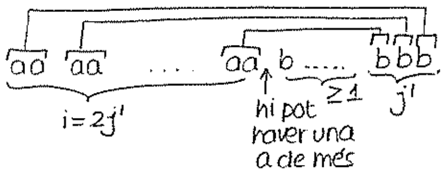
↓ les a's que afegim

↙ Té un nombre parell d'as. ara per cada 2 a's ens asseourem que hi hagi una b. I llavors afegim b's per fer el menor estricte → NO CORRECTE!

⇓ si afegim $S' \to a$ ja podem fer que existeixin paraules amb 1a1 senars.

- Fer-ho a partir de la forma general de les paraules:

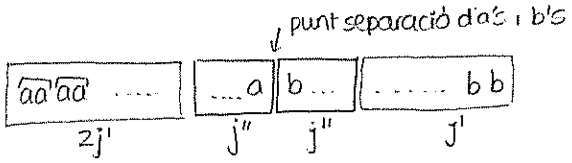




= Cal JUSTIFICAR que:
- Es generen totes les paraules del llenguatge
- Que no ens en deixem cap
- Ambigüetat

→ (g) $\{a^ib^j \mid i>j \lor i<2j\} = \{a^ib^j \mid i>j\} \cup \{a^ib^j \mid i<2j\}$

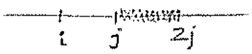
- $S \to S_G \mid S_F$ ⇒ Es ambigua perquè alguns els podem generar tant en S_G com S_F ja que són llenguatges NO disjunts ($S_G \cap S_F \neq \emptyset$).



⇓

són totes les paraules $\{a^ib^j \mid i\neq 0 \land j\neq 0\}$ ⇒ la propa definició ja és ambigua.

⇒ $\{a,b\}^* - \lambda$



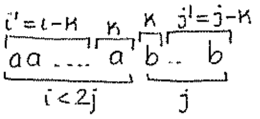
$S \to aS \mid Sb \mid a \mid b$ Ambigu

$S \to aAB \mid ABb$
$A \to aA \mid \lambda$
$B \to bB \mid \lambda$
} NO ambigu.

− (h) $\{a^ib^j \mid j<i<2j\}$

- $S \to aaSb \mid aS'b$
  $S' \to aS'b \mid ab$

↓ gramàtica NO ambigua.



per cada b també treiem 1 a. Però en l'expressió eliminar 1i és eliminar 2j's

$i>j$
$i<2j$
$(-1) \Rightarrow (-2)$
$\boxed{i' = 2j'}$ sobre un cert j'

11 Abril 05

(j) $\{a^ib^jc^k \mid i=j+k\} \equiv \{a^ka^jb^jc^k \mid k,j \geq 0\}$

$S \to aSc \mid S'$
$S' \to aS'b \mid \lambda$

(k) $\{a^ib^jc^k \mid i \geq j+k\} \equiv \{a^\alpha a^ka^jb^jc^k \mid \alpha \geq 1,\ k,j \geq 0\}$

$S \to aSc \mid S'$
$S' \to aS'b \mid A$
$A \to Aa \mid \lambda$.

(l) $\{a^ib^jc^k \mid 2i=j+k\}$

$S \to aScc \mid S' \mid aS'bc$
$S' \to aS'bb \mid \lambda$

volem demostrar $\begin{cases} L(S)=\{a^ib^jc^k \mid 2i=j+k\} \\ L(S')=\{a^ib^{2i}\} \end{cases}$

justifiquem primer:

$L(S) \subseteq \{a^ib^jc^k \mid 2i=j+k\}$
$L(S') \subseteq \{a^ib^{2i}\}$

Ho demostrem per inducció sobre el nombre de passos de reescriptura des de S o S'.

Considerem una derivació qualsevol $\left\{\begin{matrix} S \to^* w \\ \text{o} \\ S' \to^* w \end{matrix}\right\}$ on w és paraula final.

Considerem casos en funció del primer pas de reescriptura:

(a) $S \to aScc \xrightarrow{*} aw'cc = w$ on $S \xrightarrow{*} w'$ és una subderivació de l'anterior amb menys passos de reescriptura, i per HI, $w'=a^ib^jc^k$ amb $2i=j+k$

Per tant, $w=aw'cc=aa^ib^jc^kcc = a^{i+1}b^jc^{k+2}$ i es compleix $2(i+1)=j+k+2$,
$2i+\cancel{2}=j+k+\cancel{2}$ ✓

(b) $S \to S' \xrightarrow{*} w$ on la subderivació $S' \to^* w$ té menys passos de reescriptura i per tant per HI $w=a^ib^{2i}$ que pertany a $\{a^ib^jc^k \mid 2i=j+k\}$

(c) $a \to aS'bc \xrightarrow{*} aw'bc = w$ on la subderivació $S' \to^* w'$ té menys passos de reescriptura.

Per HI, $w'=a^ib^{2i}$, i per tant $w=aw'bc=aa^ib^{2i}bc=a^{i+1}b^{2i+1}c$, i es compleix $2(i+1)=2i+1+1$

(d) $S' \to aS'bb \xrightarrow{*} aw'bb = w$ on $S' \to w'$ té menys passos i per HI $w'=a^ib^{2i}$, $w=aw'bb=a^{i+1}b^{2i+2}=a^{i+1}b^{2(i+1)}$

(e) $S' \to \lambda = w$, i $\lambda = a^0b^{2\cdot 0}$

Justifiquem primer $L(S') \supseteq \{a^ib^{2i}\}$. Ho fem per inducció sobre la mida de la paraula. $w=a^ib^{2i}$

(a) si $|w|=0$ llavors $w=\lambda$, però tenim la regla $S' \to \lambda$.

(b) si $|w|>0$

llavors $i>0$, i $w=a^ib^{2i}=aa^{(i-1)}b^{2(i-1)}bb$. La paraula $a^{(i-1)}b^{2(i-1)}$ també és del llenguatge, i té menys mida. Per HI $S' \xrightarrow{*} a^{(i-1)}b^{2(i-1)}$ i llavors també tinc la derivació:

Ara el que falta per veure $L(S) \supseteq \{a^ib^jc^k \mid 2i=j+k\}$

Ho veiem un altre cop per inducció sobre la mida de $w=a^ib^jc^k$

(a) si la paraula w no té c's llavors és de la forma: $w=a^ib^{2i}$, i amb la demostració anterior obtenim $S \to S' \to^* a^ib^{i2}=w$

(b) si la paraula w té exactament una sola c, llavors $i>0$, $j>0$ $k=1$, i w es de la forma $w=a^ib^jc=aa^{(i-1)}b^{(j-1)}bc$ i donat que $2i=j+k=j+1$ s'ha de complir $2(i-1)=(j-1)$ La paraula $a^{(i-1)}b^{j-1}=$ $=a^{i-1}b^{2(i-1)}$ com hem demostrat abans, és generable des de S', i llavors tenim la derivació $S \rightarrow aS'bc \xrightarrow{*} a\,a^{(i-1)}b^{(j-1)}bc = \omega$

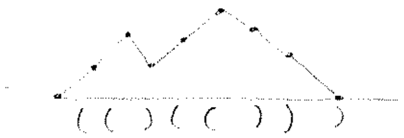
(c) si la paraula ω té 2 o + c's, llavors $k \geq 2$, $i > 0$ i la paraula $\Rightarrow \omega = a^i b^j c^k = a a^{i-1} b^j c^{k-2} cc$, i la subparaula $a^{i-1}b^j c^{k-2}$ és del llenguatge, doncs es compleix $2(i-1) = j+k-2$, ja que $2i = j+k$, i és més petita i per HI: $S \xrightarrow{*} a^{i-1}b^j c^{k-2}$ i també tenim la derivació $S \rightarrow aScc \xrightarrow{*} a a^{(i-1)} b^j c^{k-2} cc = \omega$.

– * EXERCICI 1:

- Intenten donar una definició amb notació de conjunts de les seqüències ben parantitzades i també una gramàtica que les generi.

$\{\omega \in \{(,)\}^* \mid |\omega|_(= |\omega|_) \} \wedge \forall x,y : (\omega = \overset{\text{prefix}}{x}y \Rightarrow |x|_(\geq |x|_)$ $\quad |x|_(- |x|_) \geq 0$

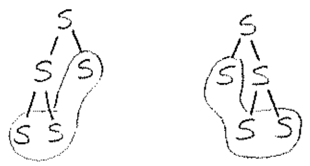
Fer gramàtica per a seq. ben parantitzades amb 2 tipus de parèntesis () i []



$\{\omega \in \{(,)\}^* \mid |\omega|_(= |\omega|_) \quad \wedge (\forall x,y : \ \omega = xy \Rightarrow |x|_(\geq |x|_))\}$

↑ tot prefixe x de ω compleix

$\Rightarrow S \rightarrow (S) \mid \lambda \mid SS$ és ambigua



$\Rightarrow S \rightarrow (S)S \mid \lambda$

$\equiv$

$S \rightarrow S(S) \mid \lambda$

= * Exercici 2: Fer gramàtica per a seq. ben parentitzades amb 2 tipus de parèntesis ( ) i [ ].

$\{\omega \in \{(,),[,]\}^* \mid \forall xyz : (\omega = x(y \Rightarrow (\exists y',z : y = y')z \wedge |y'|_(= |y|_) \wedge |y'|_[= |y|_])) \wedge (\omega = x[y \Rightarrow (\exists y',z : y = y']z \wedge |y'|_(= |y|_) \wedge |y'|_[= |y|_])))\}$

$S \rightarrow (S)S \mid [S]S \mid \lambda$

= * Exercici 3:

- $S \rightarrow (S \mid (S)S \mid \lambda$ és ambigua
- $S \rightarrow (S \mid (X)S \mid \lambda$
  $X \rightarrow (X)X \mid \lambda$ ← Genera de forma ben parantitzada.

* Exercici 4:

- $S \rightarrow aSbS \mid bSaS \mid \lambda$ és ambigua
- $S \rightarrow aAbS \mid bBaS \mid \lambda$
  $A \rightarrow aAbA \mid \lambda$
  $B \rightarrow bBaB \mid \lambda$

* Exercici 5:

$S \rightarrow aAbS \mid bAaS \mid \lambda \mid aA'$
$A \rightarrow aAbA \mid \lambda$
$B \rightarrow bBaB \mid \lambda$
$A' \rightarrow aAbA' \mid aA' \mid \lambda$

15 Abril 05

Llenguatges incontextuals ≡ els generables per gramàtiques

Una gramàtica és una tupla $< \Sigma, V, P, S > = G$

Σ : símbols terminals
V : símbols de variable
P : conjunt de regles o produccions $\subseteq V \times (V \cup \Sigma)^*$ → possibilitat de substituir una variable per cadenes de variables
$S \in V$: símbol inicial.

$L(G)$ → Llenguatge generat per G, són les paraules de Σ^* accessibles desde S utilitzant P com a sistema de reescriptura.
$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$

Els llenguatges incontextuals són tancats per:

- Unió (o reunió), és a dir, si L_1, L_2 són incontextuals, llavors $L_1 \cup L_2$ també ho és.

  Ex: La + és una op tancada per als $\mathbb{N}$
  La / no és una op tancada per als $\mathbb{N}$ ← El 0 no està definit.

  Si G_1 és una gramàtica de L_1 i G_2 és una gramàtica de L_2 i les hem construit de manera que no comparteixen variables, llavors:
  $G_1 \left[S_1 \to \dots \right]$ $G_2 \left[S_2 \to \dots \right]$

  $$\left[\begin{array}{l} S \to S_1 \mid S_2 \\ S_1 \to \dots \\ \dots \\ S_2 \to \dots \\ \dots \end{array}\right]$$

  és una gramàtica per $L_1 \cup L_2$ i per tant, $L_1 \cup L_2$ és incontextual

- També són tancats per concatenació

  $L_1 \cdot L_2 = \{ w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2 \}$

  si L_1 té gramàtica $G_1 \left[S_1 \to \dots \right]$ i L_2 té gramàtica $G_2 \left[S_2 \to \dots \right]$, llavors $L_1 \cdot L_2$ té gramàtica $\left[\begin{array}{l} S \to S_1 \cdot S_2 \\ S_1 \to \dots \\ \dots \\ S_2 \to \dots \\ \dots \end{array}\right]$

- També són tancats per l'operació * (Tancament de Kleene)
  Totes les possibles concatenacions de les paraules d'un llenguatge

  $L^* = \{ w_1 \dots w_n \mid n \geq 0 \wedge \forall i \in \{1..n\} : w_i \in L \}$ $\quad L^* = \bigcup_{n \geq 0} L^n$
  $= \{\lambda\} \cup L \cup LL \cup LLL \cup \dots =$
  $= L^0 \cup L^1 \cup L^2 \cup L^3 \dots$

  $L^n = \underbrace{L \cdot L \cdot \dots \cdot L}_{n}$

  El neutre de la concatenació de llenguatges és el llenguatge $\{\lambda\}$.

  $L \cdot \phi = \phi$; $L \cdot \{\lambda\} = L$ $\qquad$ Tancament Positiu: $L^+ = \bigcup_{n > 0} L^n$

  si L té gramàtica $G \left[S \to \dots \right]$

  L^* té gramàtica $\left[\begin{array}{l} S' \to SS' \mid \lambda \\ S \to \dots \\ \dots \end{array}\right]$ $\qquad$ L^+ té gramàtica $\left[\begin{array}{l} S' \to SS' \mid S \\ S \to \dots \\ \dots \end{array}\right]$

- També són tancats sota el revessat

  $L^R = \{ w^R \mid w \in L \}$
  $w^R = (a_1 a_2 \dots a_n)^R = a_n a_{n-1} \dots a_2 a_1$

  si L té gramàtica $\left[\begin{array}{l} S \to u_1 \mid u_2 \mid \dots \\ X \to v_1 \mid v_2 \mid \dots \\ Y \to w_1 \mid w_2 \mid \dots \end{array}\right]$

L^R té gramàtica $\left[\begin{array}{l} S \to u_1^R \mid u_2^R \mid \dots \\ X \to v_1^R \mid v_2^R \mid \dots \\ y \to w_1^R \mid w_2^R \mid \dots \end{array}\right]$

es pot justificar amb la propietat $(w_1 w_2)^R = w_2^R \cdot w_1^R$

$(w_1 \dots w_n)^R = w_n^R \, w_{n-1}^R \dots w_1^R$

⇒ No són tancats per <u>intersecció</u>.

Ex: $\left.\begin{array}{l} L_1 = \{a^n b^n c^n\} \\ L_2 = \{a^i b^j c^j\} \end{array}\right\}$ són incontextuals

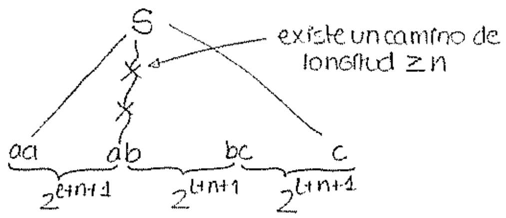
$L_1 \subseteq L_2$ en que la intersecció és el mateix L_1

$L_1 \cap L_2 = \{a^n b^n c^n\}$ no és incontextual (No existeix una gramàtica que el generi)

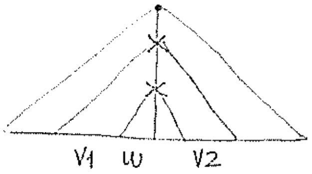
* <u>Idea Demostració</u>:

Suposem que $L_1 \cap L_2$ és incontextual.

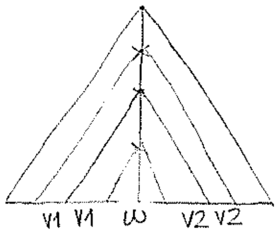
Per tant, existeix una Gramàtica G amb un cert nombre n fixe de variables, i una longitud maximal de part dreta de la regla. Agafem la paraula $a^{2^{l+n+1}} b^{2^{l+n+1}} c^{2^{l+n+1}}$ que és del llenguatge. Existeix un arbre de derivació:



En aquest camí, una variable apareix 2cops.



Es pot veure que també es genera:



que no serà del llenguatge.

El llenguatge $\{ww\}$ tampoc és incontextual, excepte si el corresponent alfabet té un sol símbol.

$\Sigma = \{a\}$, $S \to aaS \mid \lambda$ ó $S \to aSa \mid \lambda$

El llenguatge $\{ww^R\}$ sí és incontextual per qualsevol alfabet.

$S \to aSa \mid bSb \mid \lambda$

⇒ Tampoc són tancats per <u>complementari</u>.

Si ho fossin, llavors també ho serien per intersecció, doncs

$L_1 \cap L_2 \equiv \overline{\overline{L_1} \cup \overline{L_2}}$

$\overline{\{a^n b^n c^n\}}$ sí és incontextual.

- També són tancats per morfisme directe

  un morfisme σ és una funció de paraules a paraules $\sigma : \Sigma_1^* \rightarrow \Sigma_2^*$, que es defineix de la forma més senzilla possible, només cal definir la imatge dels símbols de Σ_1, i la resta queda definit per extensió a la concatenació $\sigma(w_1 w_2) = \sigma(w_1)\sigma(w_2)$.

  un cop he definit
  $$\begin{matrix} \sigma(a_1) = w_1 \\ \sigma(a_2) = w_2 \\ \vdots \\ \sigma(a_n) = w_n \end{matrix} \quad \text{on } \Sigma_1 = \{a_1 \dots a_n\}$$

  llavors: $\sigma(a_{i_1} \dots a_{i_K}) \underset{\substack{\uparrow \\ i_1 \dots i_K \in \{1 \dots n\}}}{=} \sigma(a_{i_1})\sigma(a_{i_2}) \dots \sigma(a_{i_K})$

  Exemples:

  - si definim $\sigma(a) = a$, $\sigma(b) = ab$

    $\sigma(abba) = aababa$

  - si definim $\sigma(a) = a$, $\sigma(b) = a$

    $\sigma(\{a^n b^n\}) = \{a^{2n}\} = \sigma(w \in \{a,b\}^* \mid |w|_a = |w|_b)$

  si L té gramàtica
  $$\begin{bmatrix} S \rightarrow w_1 \mid w_2 \mid \dots \\ X \rightarrow v_1 \mid v_2 \mid \dots \end{bmatrix}$$

  i tenim un morfisme $\sigma : \underset{\substack{\uparrow \\ \text{alfabet de terminals} \\ \text{de gramàtica.}}}{\Sigma_1^*} \rightarrow \Sigma_2^*$

  Per generar la nova gramàtica, primer extenem σ a variables, segons la identitat, és a dir, $\sigma(X) = X \quad \forall x \in V$ i la gramàtica que genera $\sigma(L)$ és:
  $$\begin{bmatrix} S \rightarrow \sigma(w_1) \mid \sigma(w_2) \mid \dots \\ X \rightarrow \sigma(v_1) \mid \sigma(v_2) \mid \dots \end{bmatrix}$$

- Demostració que NO són tancats respecte el complementari:
  (Reducció al Absurd)

  $\Rightarrow$ suposem que la complementació és tancada.

  $\Leftrightarrow$ suposem que $L \in CFL \Rightarrow \overline{L} \in CFL$

  siguin $L_1, L_2 \in CFL$

  $$L_1 \cap L_2 \underset{\substack{\uparrow \\ \text{lleis de} \\ \text{Morgan}}}{\equiv} \overline{\overline{L_1} \cup \overline{L_2}}$$

  $$\overline{L_1}, \overline{L_2} \in CFL \underset{\substack{\uparrow \\ \text{la unió} \\ \text{és tancada}}}{\Rightarrow} (\overline{L_1} \cup \overline{L_2}) \in CFL \underset{\substack{\uparrow \\ \text{per la} \\ \text{suposició}}}{\Rightarrow} \overline{\overline{L_1} \cup \overline{L_2}} \in CFL \not\Leftrightarrow \underbrace{\overline{\overline{L_1} \cup \overline{L_2}} \equiv L_1 \cap L_2 \notin CFL}_{\text{contradicció!!}}$$

18Abril05

Un <u>morfisme</u> és una funció $\sigma : \Sigma_1^* \rightarrow \Sigma_2^*$ que satisfà $\sigma(w_1w_2) = \sigma(w_1)\sigma(w_2)$.
Alternativament, és una funció que es pot definir únicament per Σ_1 i, automàticament queda definida per tot Σ_1^* per extensió a concatenació.

$$\sigma(a_1 \ldots a_n) = \sigma(a_1)\sigma(a_2) \cdots \sigma(a_n)$$

Exemple:

són morfismes les següents definicions?

(1) $\sigma(w) = w^R$
No és morfisme.
$\underbrace{\sigma(ab)}_{\hookrightarrow \neq \sigma(a)\sigma(b) = ab} = ba$

(2) $\sigma(a_1 \ldots a_n) = a_1a_1a_2a_2 \ldots a_na_n$
sí és morfisme
$\sigma(a) = aa \quad \forall a \in \Sigma_1$

(3) $\sigma(w) = a^{|w|}$
sí és morfisme
$\sigma(b) = a \quad \forall b \in \Sigma_1$.

(4) $\sigma(w) = ww$
No és morfisme.
$\underbrace{\sigma(ab)}_{\hookrightarrow \neq \sigma(a)\sigma(b) = aabb} = abab$

Donat un morfisme $\sigma : \Sigma_1^* \rightarrow \Sigma_2^*$

$$\begin{cases} \sigma(L) = \{\sigma(w) \mid w \in L\} \\ \sigma(L_1L_2) = \sigma(L_1)\sigma(L_2) \end{cases}$$

⇒ Demostració:

$\subseteq$) si $w \in \sigma(L_1L_2)$, llavors existeix $y \in L_1L_2 : \sigma(y) = w$, i per tant $y = y_1y_2$ on $y_1 \in L_1$, $y_2 \in L_2$. $w = \sigma(y) = \sigma(y_1y_2)$ $= \sigma(y_1)\sigma(y_2)$ i tenim que $\sigma(y_1) \in \sigma(L_1)$, $\sigma(y_2) \in \sigma(L_2)$ i finalment $w = \sigma(y_1)\sigma(y_2) \in \sigma(L_1)\sigma(L_2)$

$\supseteq$) si $w \in \sigma(L_1)\sigma(L_2)$, llavors $w = w_1w_2$ on $w_1 \in \sigma(L_1)$ $w_2 \in \sigma(L_2)$, i per tant existeixen $y_1 \in L_1$, $y_2 \in L_2$ tals que $\sigma(y_1) = w_1$, $\sigma(y_2) = w_2$
Així doncs, $y_1y_2 \in L_1L_2$, i $\sigma(y_1y_2) \in \sigma(L_1L_2)$
$\Downarrow$
per ser
σ polimorfisme
$\sigma(y_1)\sigma(y_2) \Rightarrow w_1w_2 = w$

- $\sigma(L^n) = \sigma(L)^n$

ho veiem:

$$\sigma(L^n) = \sigma(\overbrace{L \cdot \underbrace{(L \cdots L)}_{n-1}}^{n}) = \sigma(L)\sigma(\underbrace{L \cdots L}_{n-1}) = \sigma(L)\sigma(L)\sigma(\underbrace{L \cdots L}_{n-2})$$

$$= \ldots = \sigma(L)\sigma(L) \ldots \sigma(L) = \sigma(L)^n$$

- $\sigma(L^*) = \sigma(L)^* \longrightarrow L^* = \underset{\{\lambda\}}{L^0} \cup L^1 \cup L^2 \cup \ldots$

$$\sigma(L^*) = \sigma(L^0 \cup L^1 \cup L^2 \cup \ldots) = \sigma(L^0) \cup \sigma(L^1) \cup \ldots = \sigma(L)^0 \cup \sigma(L)^1 \cup \sigma(L)^2 \cup \ldots =$$

$$= (\sigma(L))^*$$

Els llenguatges incontextuals són tancats per morfisme directe, és a dir, si L té gramàtica que el genera, i σ es morfisme, llavors σ(L) també és una gramàtica associada.

També són tancats pes morfisme invers, és a dir, si L és incontextual, llavors $\sigma^{-1}(L)$ també ho és, per qualsevol morfisme σ. → Pendent a justificar.

EXEMPLES:

$L1 = \{a^n b^n\}$
$L2 = \{ w \in \{a,b\}^* \mid |w|a = |w|b \}$

$\sigma(a) = a$
$\sigma(b) = a$

longitud parell
$\sigma(L1) = \{a^{2i}\} = \sigma(L2)$ El morfisme pertany a un altre llenguatge.

antimatge en sentit de conjunts
$\sigma^{-1}(\{a^{2i}\}) = \{ w \in \{a,b\}^* \mid |w| = \dot{2} \}$

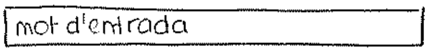
Gramàtica per L1 ⇒ $G1 = S \rightarrow aSb \mid \lambda$
Gramàtica per L2 ⇒ $G2 = S \rightarrow aSbS \mid bSaS \mid \lambda$

$\sigma(L1) = G1' \rightarrow S \rightarrow \underset{\sigma(a)=a}{a}S\underset{\sigma(b)=a}{a} \mid \lambda$
$\sigma(L2) = G2' \rightarrow S \rightarrow aSaS \mid aSaS \mid \lambda$

• Autòmats Amb Pila. (Indeterministes)

PDA (Push-Down automaton) Model acceptador No Genera. RESTRICCIONS:

Entrada de mida fixa i no modificable. → llegim el símbol actual però no el podem canviar.



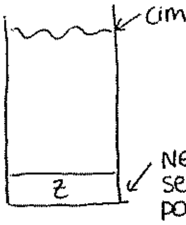
Existeix un capçal mòbil i a cada moviment només poden recordar certa informació.

Pila

Es pot escriure, llegir i modificar el que hi ha al cim de la pila

memòria on emmagatzema símbols de treball

⇒ Permet fer comparacions i comptatges sobre els mots d'entrada.

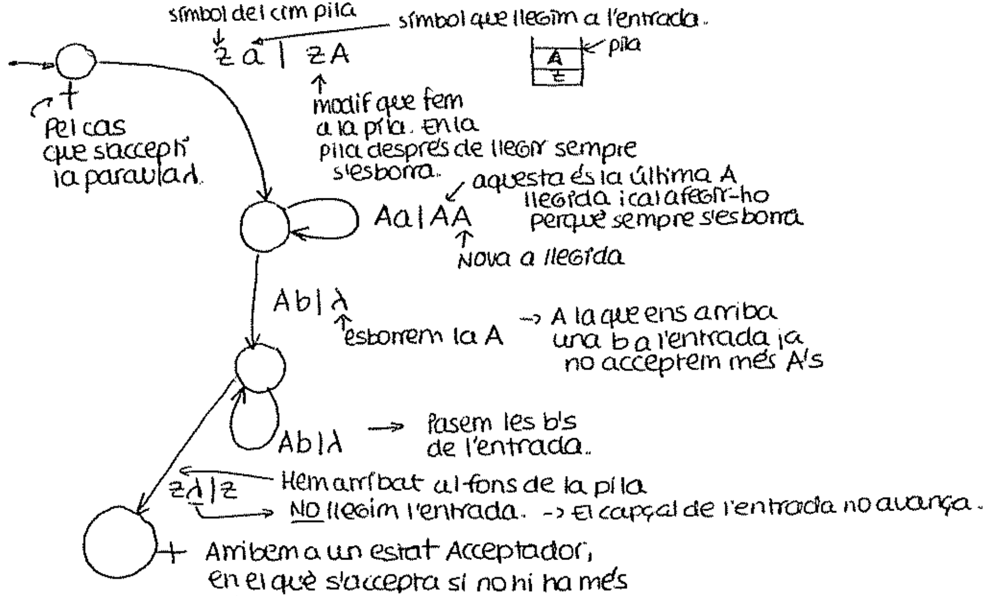


Necessitem una senyal que l'autòmat pogués llegir que ens informi que la pila està buida ⇒ z

- l'Entrada es llegeix d'esquerra a dreta (→) i només així.
- s'accepta indeterminisme
- Decideix si accepta l'entrada o no

(*) En un moment determinat hi ha 2 transicions per executar i se n'escull una i s'accepta l'entrada. si cap execució accepta l'entrada llavors es diu que es rebutja.

Exemple: Autòmat amb pila (PDA) pel llenguatge $\{a^n b^n\}$



símbol del cim pila

símbol que llegim a l'entrada.

modif que fem a la pila. En la pila després de llegir sempre s'esborra.

Pel cas que s'accepti la paraula λ.

aquesta és la última A llegida i cal afegir-ho perquè sempre s'esborra

Nova a llegida

esborrem la A → A la que ens arriba una b a l'entrada ja no acceptem més A's

→ Pasem les b's de l'entrada.

Hem arribat al fons de la pila

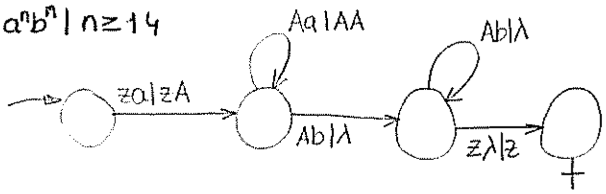
NO llegim l'entrada. → El capçal de l'entrada no avança.

Arribem a un estat Acceptador, en el què s'accepta si no hi ha més símbols a llegir en l'entrada.

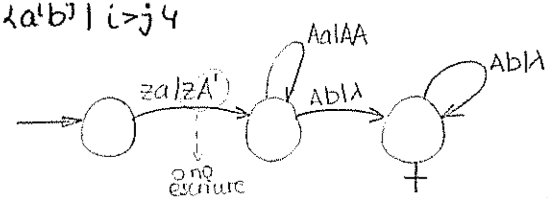
aabbb → Es rebutja la paraula perquè no hi ha trans. definida, encara tenim símbols a llegir

- Exercici ⇒ fer autòmats amb pila per:
  - (1) $\{a^n b^n \mid n \geq 1\}$
  - (2) $\{a^i b^j \mid i > j\}$
  - (3) $\{a^i b^j \mid i < j\}$
  - (4) $\{a^i b^j \mid i > 2j\}$
  - (5) $\{a^i b^j \mid i < 2j\}$

(1) $\{a^n b^n \mid n \geq 1\}$



(2) $\{a^i b^j \mid i > j\}$



deixem una 'a' de diferència amb les b's. (Tenim segur una a de més)

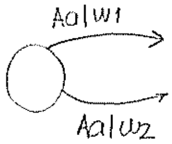
- Conceptes (a nivell intuïtiu):

  En general indeterminisme ≡ existència de camins d'execució diferents. Llavors diem que acceptem una entrada si existeix un camí d'execució acceptador.

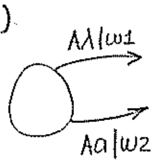
  En general determinisme ≡ no indeterminisme, és a dir, equival a l'existència d'un únic camí d'execució per cada entrada.

  Pel cas d'autòmats amb pila, això no és exacte.
  Diem que un autòmat amb pila és determinista si:

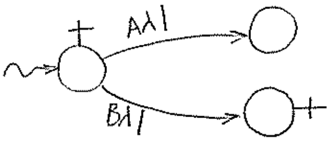
  - No tinc transicions amb condicions duplicades



  - A més, si hi ha definida una λ-transició amb un cert símbol A de pila, llavors no hi ha cap transició definida amb A i fent una lectura (això ha de passar per cada estat).



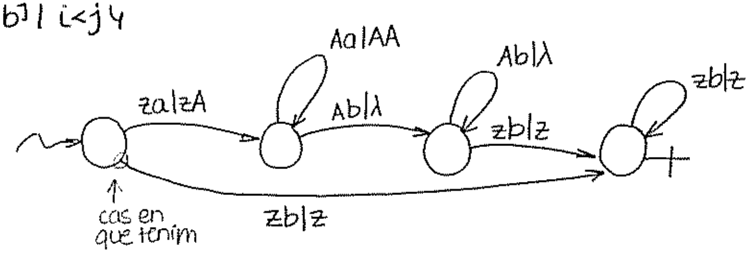
Un PDA (Autòmat amb pila) determinista pot no tenir camins d'execució únics (ni tan sols un únic camí d'acceptació en cas que la paraula s'accepti). És degut a aquesta situació anòmala:



Poden existir 2 o més exec. acceptadores.

Autòmats amb pila d'acceptació única: si accepta una entrada, llavors hi ha un únic camí d'execució acceptador.
Tot autòmat determinista no es pot transformar en un de determinista i d'acceptació única.

(3) $\{a^ib^j \mid i<j\}$

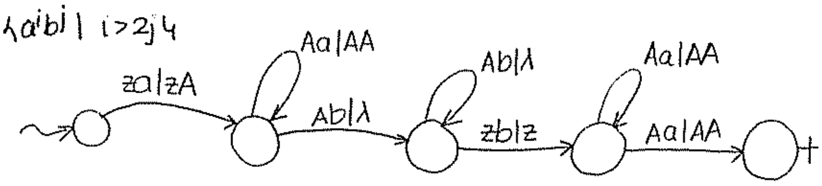


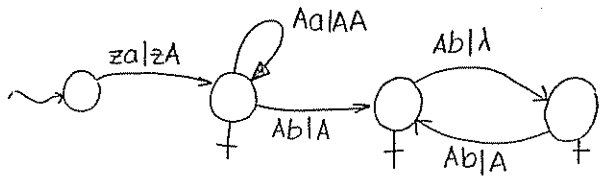
cas en que tenim 0 a's i hem de contar b's

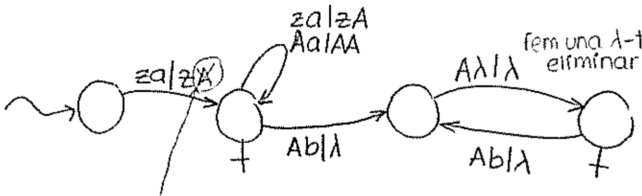
Autòmat amb 2 piles és indecidible perquè és fàcil simular la cinta d'una màquina de Turing => pot calcular qualsevol cosa. Si fos una cua també es pot convertir en una màq. de Turing i es pot computar tot. a^nb^n, no es pot reconèixer i amb TM sí.

Qualsevol cosa decidible es pot resoldre utilitzant dues piles.

(4) $\{a^ib^j \mid i>2j\}$





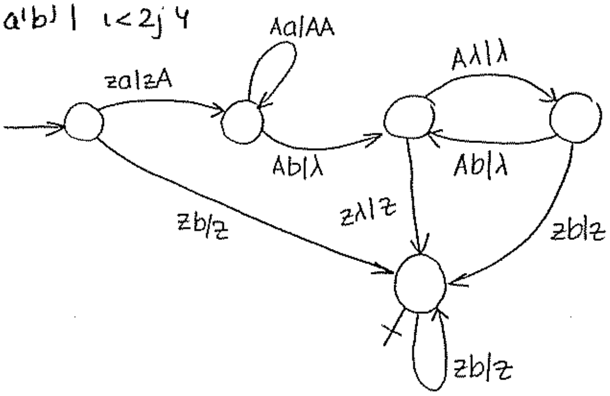


fem una λ-transició per eliminar per cada b dues A's

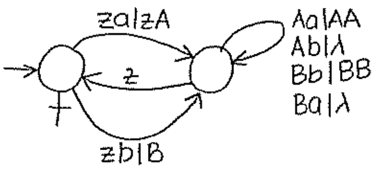
Hem passat per una A sense contar-la així assegurem i(>)2j

Estaríem reconeixent i ≥ 2j
Posar totes les A's i quan trobo B's esborrem 2 A's

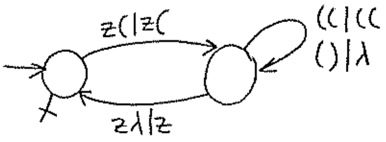
(5) $\{a^ib^j \mid i<2j\}$



(7) $\{w\in\{a,b\}^* \mid |w|_a=|w|_b\}$



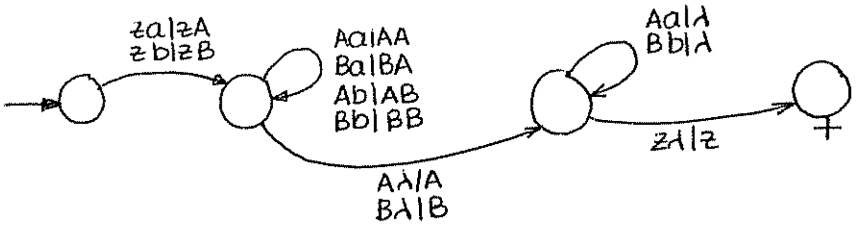
(8). Ben parentitzats



22Abril05.

Aquest llenguatge és reconeixible amb un autòmat amb pila?

(9) • $\{ww^R \mid w \in \{a,b\}^*\}$

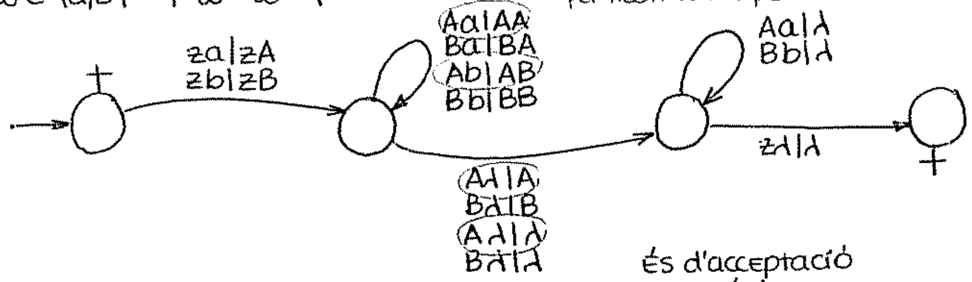


↑
situació d'indeterminisme
No existeix cap màq. determinista que reconegui aquest llenguatge, ja que no hi ha forma de saber la meitat.

Però és d'acceptació única. Tota paraula acceptada té una única execució acceptadora: la que escull fer una λ-transició exactament quan arribem a la meitat de la paraula.

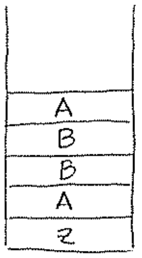
(10) • $\{w \in \{a,b\}^* \mid w = w^R\}$

Hi ha 4 entrades definides per llegir a i 4 per llegir B



És d'acceptació única.

abbba



No accepta, en executar aquest punt rebutja perquè no hi ha elements per llegir

(11) • $\{a^ib^jc^k \mid i=j \vee i=k\}$

Existeix un model alternatiu on l'autòmat és determinista, i podem moure el capçal esq. i dreta (i més detalls) que s'anomenen:

2DPDA {
D: Determinista
PDA: Autòmat amb pila
2: Bidireccional

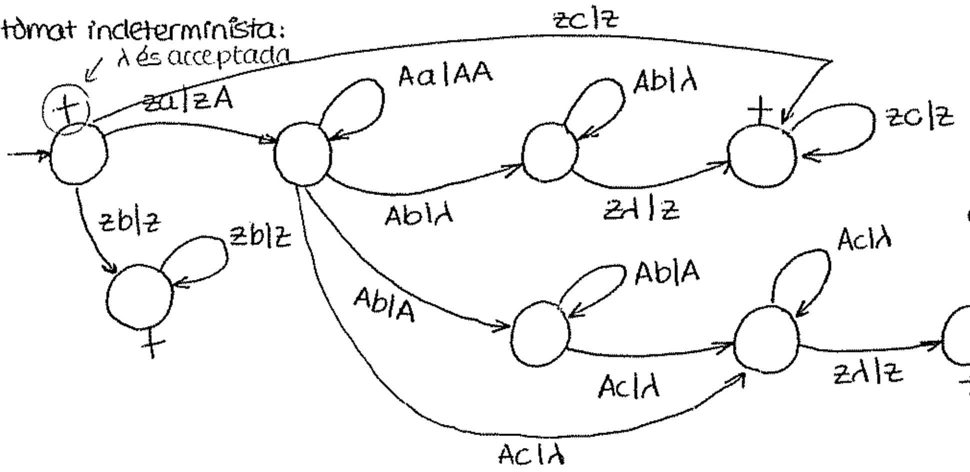
i que permeten reconèixer $\{a^nb^nc^n\}$ que no és incontextual 2DPDA $\not\subseteq$ CFL

(CFL: llenguatges acceptats per CFG's) gramàtiques,

Curiositat: Mai ningú ha aconseguit demostrar CFL $\subseteq$ 2DPDA.

Autòmat indeterminista:

λ és acceptada



No és d'acceptació única.
Hi han 2 execucions acceptadores per abc.

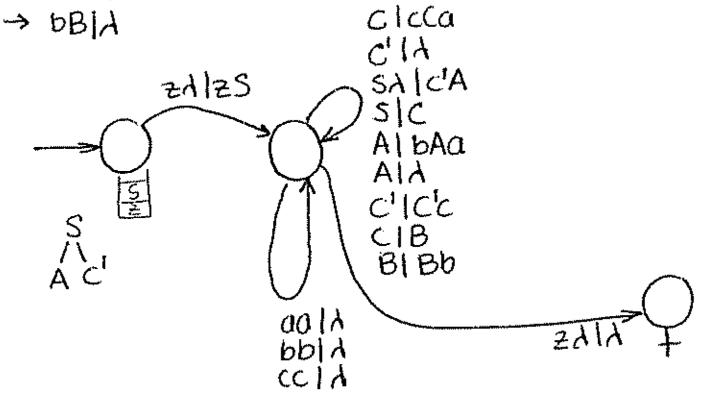
Mateix llenguatge anterior: $\{ a^i b^j c^k \mid i=j \vee j=k \}$

És inherentment ambigua: No existeix cap gramàtica inambigua que el reconegui.

Gramàtica pel llenguatge:

$S \rightarrow AC' \mid C$
$A \rightarrow aAb \mid \lambda$
$C' \rightarrow cC' \mid \lambda$
$C \rightarrow aCc \mid B$
$B \rightarrow bB \mid \lambda$

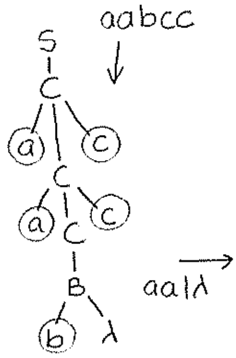
Generem el corresponent autòmat amb pila, que bàsicament, simularà tots els àrbres de derivació de la gramàtica.



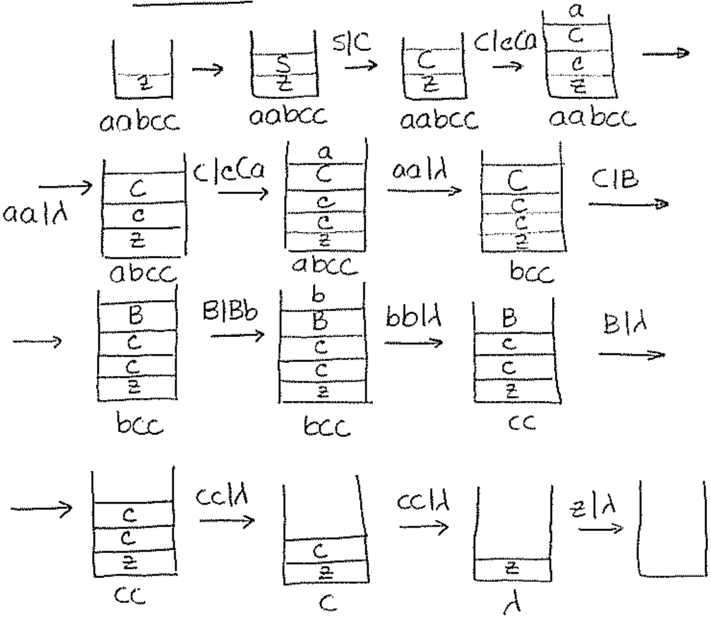
Reconeix el mateix llenguatge que l'autòmat anterior però a partir de la generació de la gramàtica del llenguatge.

Execcució de l'autòmat amb la paraula del llenguatge aabcc.

Arbre de derivació:



Execució:



! Qualsevol llenguatge incontextual és reconeixible per un autòmat amb pila indeterminista.

A més, donat que hi ha una correspondència unívoca entre arbres de derivació de paraules terminals i execucions acceptadores, si la gramàtica és no ambigua, llavors l'autòmat és d'ACCEPTACIÓ ÚNICA.

L'antiimatge d'un llenguatge incontextual, és incontextual

$L \in CFL$ i σ morfisme $\Rightarrow \sigma^{-1}(L) \in CFL$

1 Transformació de PDA's en CFG's

Veiem a continuació com construïr una gramàtica que generi el llenguatge reconegut per un autòmat amb pila donat. Aquesta construcció satisfà que, si l'autòmat inicial d'acceptació única, és a dir, que tota paraula acceptada tingui una única execució acceptadora, llavors la gramàtica final és inambigua (tot i que és possible que s'hagin generat molts símbols no útils, que es podrien eliminar posteriorment, o en la mateixa construcció fent un preprocés adequat).

Suposem un cert autòmat amb pila donat, que essencialment consta d'un conjunt d'estats als que ens referirem amb la notació q o q_i, i d'unes transicions que denotarem com $q_i \rightarrow_{Aa|w} q_j$.

Construïm les següents variables per a la gramàtica:

- $[A, q_1, q_2]$ generarà totes les paraules w tals que existeix una execució en l'autòmat desde l'estat q_1 amb A en la pila que acaba en l'estat q_2 havent adquirit alçada 0 només en l'últim pas d'execució.
- $< A, q_1, q_2 >$ genera totes les paraules w tals que existeix una execució en l'autòmat desde l'estat q_1 amb A en la pila que acaba en l'estat q_2, i on la pila mai adquireix alçada 0 (i on potser s'acaba amb encara més símbols que 1 sobre la pila).

Definim la gramàtica:

- Per cada transició $q_1 \rightarrow_{Aa|\lambda} q_2$ afegim la regla $[A, q_1, q_2] \rightarrow a$, i, anàlogament, per cada transició $q_1 \rightarrow_{A\lambda|\lambda} q_2$ afegim $[A, q_1, q_2] \rightarrow \lambda$.
- Per cada transició $q_0 \rightarrow_{Aa|A_n \ldots A_1} q_1$ amb $n \geq 1$ i totes les possibles eleccions d'estats $q_2 \ldots q_n, q_{n+1}$ afegim la regla

  $$[A, q_0, q_{n+1}] \rightarrow a[A_1, q_1, q_2][A_2, q_2, q_3] \ldots [A_n, q_n, q_{n+1}]$$

  anem esborrant els cims

  I també totes les possibles regles de la forma

  $$< A, q_0, q_{i+2} > \rightarrow a[A_1, q_1, q_2] \ldots [A_i, q_i, q_{i+1}] < A_{i+1}, q_{i+1}, q_{i+2} >$$

  amb $i < n$, que intuitivament correspon al fet que els símbols $A_n \ldots A_{i+2}$ no s'accediran mai en la execució perque el símbol A_{i+1} com a molt serà modificat, però mai esborrat. (En realitat ens podríem limitar, en aquest segon cas, als q_{i+2} que siguin estat final).
- Per a qualsevol estat q i qualsevol símbol de pila A afegim

  $$< A, q, q > \rightarrow \lambda$$

  (En realitat ens podríem limitar als q que siguin estat final).
- Creem un símbol inicial S per a la gramàtica, i si Z és el símbol de fons de pila, q és l'estat inicial i $q_1 \ldots q_k$ són els estats acceptadors, afegim les regles:

  $$S \rightarrow [Z, q, q_1] \mid \ldots \mid [Z, q, q_k] \mid < Z, q, q_1 > \mid \ldots \mid < Z, q, q_k >$$

Es poden justificar les següents propietats respecte a la construcció que acabem de realitzar.

a) Es pot construïr inductivament una transformació tal que, donada una execució E sobre l'autòmat, que començarà desde un cert estat q_1 contenint només un cert símbol A a la pila i que es llegirà una paraula w portant-nos a un cert estat q_2, dona com a resultat un arbre de derivació de w de la gramàtica començant desde $[A, q_1, q_2]$ o $< A, q_1, q_2 >$.

b) Per altra banda, es pot construïr inductivament una transformació injectiva tal que, donada una variable de la gramàtica $[A, q_1, q_2]$ o $< A, q_1, q_2 >$ i una derivació esquerra desde aquesta en una paraula terminal w, dona com a resultat una execució E en l'autòmat que comença amb q_1 com a estat inicial, A a la pila, i w a la entrada, que la llegeix completament, i acaba en l'estat q_2.

Els dos punts anteriors justifiquen que l'autòmat i la gramàtica són equivalents (un reconeix el llenguatge que l'altra genera).

En el cas d'autòmats amb pila d'acceptació única, sabem que, fixada una paraula acceptadora w, existeix una única execució E començant desde l'estat inicial q, i amb Z a la pila, que es llegirà tota la paraula w i ens portarà a un cert estat final q_f. D'aquest fet se'n pot deduïr que hi ha un únic arbre de derivació de w desde el símbol inicial S. Això és conseqüència de la injectivitat de la transformació de l'apartat *b*): dos arbres de derivació diferents ens donarien lloc a dues execucions acceptadores diferents de la paraula w.

2 Transformació de PDA's deterministes en deterministes i d'acceptació única

Aquesta construcció és molt simple, i es basa en eliminar els casos anòmals. Siguin $q_1 \ldots q_n$ els estats originals. Crearem uns duplicats $q'_1 \ldots q'_n$. Per cada transició buida $q_i \rightarrow_{A\lambda|w} q_j$, afegim una nova transició $q'_i \rightarrow_{A\lambda|w} q'_j$, i per cada transició no buida $q_i \rightarrow_{Aa|w} q_j$ afegim $q'_i \rightarrow_{Aa|w} q_j$. Per cada estat final q_f i cada transició buida $q_f \rightarrow_{A\lambda|w} q_j$, la esborrem i afegim $q_f \rightarrow_{A\lambda|w} q'_j$.

És fàcil justificar que el nou autòmat continua sent determinista, però ademés, és d'acceptació única. Això és degut a que, desde estats acceptadors, ens hem assegurat que, via λ-transicions saltem a estats no acceptadors q'_i, i que només podem tornar als estats originals llegint un nou símbol de la entrada.

Qualsevol PDA té una gramàtica que genera el mateix llenguatge que l'autòmat (PDA) reconeix.

CFL: Llenguatges generats per alguna gramàtica ≡ Llenguatges reconeguts per algun autòmat amb una pila indeterminista.

A més, si el PDA és d'acceptació única, llavors la gramàtica és NO ambigua.

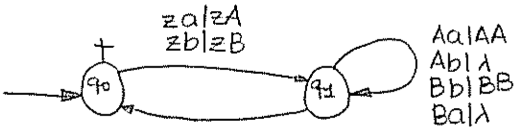
Inambigus: Classe dels llenguatges generats per una gramàtica NO ambigua ≡ Classe de llenguatges reconeguts per un autòmat amb pila.

Anomenem DCFL a la classe de llenguatges reconeguts per un autòmat amb pila determinista. (llenguatge incontextual determinista)

Els CFL són tancats per morfisme invers, doncs tota CFG té un PDA equivalent, i obtenir-ne el PDA que reconeix (antiimatge per morfisme es pot fer de manera senzilla. Aquest PDA antiimatge tindrà també una CFG equivalent.

Exemple:

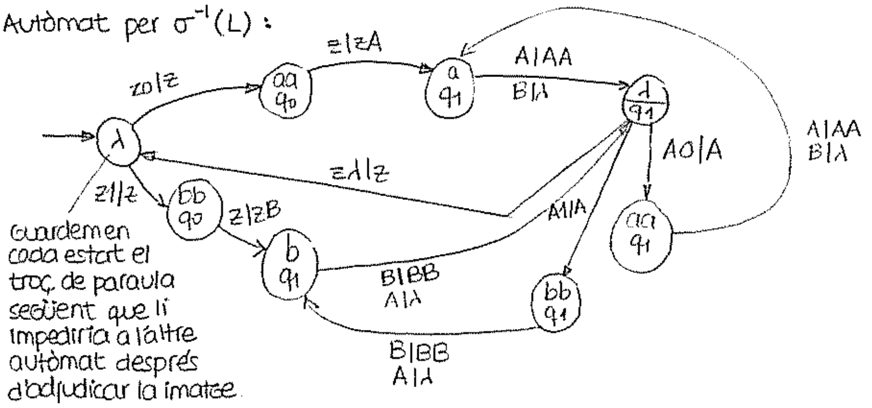
$L = \{ w \in \{a,b\}^* \mid |w|_a = |w|_b \}$



Suposem que tindrem el morfisme

$\sigma(0) = aa$
$\sigma(1) = bb$
$\sigma^{-1}(L) = \{ w \in \{0,1\}^* \mid |w|_1 = |w|_0 \}$

Autòmat per $\sigma^{-1}(L)$:



Guardem en cada estat el troç de paraula següent que li impediria a l'altre autòmat després d'adjudicar la imatge.

Posem com a acceptadors als estats que ja no eren en l'autòmat original, i tals que no els hi quedi res per llegir a l'entrada simulada. De l'exemple en deduïm que els CFL són tancats per morfisme invers, però també ho són els inambigus i els DCFL, doncs la simulació preserva el determinisme i l'acceptació única.

= Recordem que els CFL eren tancats per morfisme directe.
Això no passa pels inambigus i els DCFL

Contraexemple:

$L = \{a^n b^n c^m\} \cup \{e^k f^\ell g^\ell\}$ és DCFL.

$\sigma(a) = a \quad \sigma(c) = c \quad \sigma(f) = f \qquad \sigma(L) = \{a^i b^j c^k \mid i = j \lor i = k\}$
$\sigma(b) = b \quad \sigma(e) = e \quad \sigma(e) = \ell \qquad$ que és INHERENTMENT ambigu.

= Recordem que els CFL no eren tancats per complementari, i un contraexemple era $\{a^n b^n c^n\}$

= Els DCFL sí són tancats per complementari
Idea Bàsica: Intercanviant estats acceptadors per no acceptadors reconec exactament les paraules.
Problemes en les transicions no definides i el nostre determinisme no és determinisme real, i a més per culpa de λ-transicions pot ser que l'autòmat entri en bucle. Solució → Duplicar alguns estats.
↳ Transicions no definides
↳ El nostre determinisme admet vàries execucions degut a les λ-transicions.

↳ Hi poden haver execucions infinites degut als bucles de les λ-transicions.

Es pot transformar l'autòmat en un altre també determinista que no presenta cap d'aquests problemes i que accepta el mateix llenguatge (la transformació mirar-la a la fotocòpia).

= Recordem que els CFL eren tancats per unió.
Els DCFL i els inambigus no ho són.
$\{a^nb^nc^m\} \cup \{a^kb^lc^k\}$ és ambigu
$\nwarrow \quad \nearrow$
Els 2 són DCFL

En conseqüència, tampoc són tancats per intersecció, doncs $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$ i llavors tindrem que són tancats per unió.
Però tant, els CFL incontextuals com els inambigus i els DCFL són tancats per intersecció amb llenguatge <u>regular</u>.

Def: Un llenguatge regular és aquell que es pot reconèixer amb un PDA que no modifiqui la pila.

Donat un autòmat amb pila i un sense pila puc fer-ne un de nou amb pila que simuli alhora l'execució dels 2 anteriors.
Cada estat del nou autòmat guarda 2 estats, un per cadascún dels autòmats anteriors.
Seran estats acceptadors aquells que continguin dos estats acceptadors dels autòmats originals doncs estem reconeixent la intersecció.

Normalització de Gramàtiques:

Depuració de Gramàtiques.

<u>1ª Transformació</u>: Eliminem λ-produccions. ($X \to \lambda$)

És possible obtenir una nova gramàtica sense λ-produccions generant el mateix llenguatge (excepte per λ)

Definició: un símbol X és anul·lable per G si $X \xrightarrow[G]{*} \lambda$

Càlcul de les variables:

1. $A := \{X \mid X \to \lambda \in G\}$
   $\uparrow$
   conjunt sobre el qual calculem les var. anul·lables
2. Mentre hi hagi $X \notin A$ tal que $X \to X_1 \ldots X_n \in G$ i $X_1 \ldots X_n \in A$, fer $A := A \cup \{X\}$

Està clar que tota variable afegida a A és enumerable. Es pot demostrar per inducció sobre el nombre de passos de reescriptura $X \to \ldots \to \lambda$ que qualsevol variable anul·lable X serà afegida a A per l'algorisme.

- Eliminació de les λ-produccions. A partir de G i anul·lables (G) construïm la nova G' amb les regles següents:

El nombre de combinacions pot resultar exponencial
$X \to a\underbrace{X_1 \ldots X_n}_{anul} \to 2^n$ regles noves. Si volem algorisme de temps polinòmic no serveix

⟹ Noves regles. simulen el que es podia fer abans sense λ-produccions

$W \to aXY\,ba\,ZT$ Anul·lables(G) = {X,T}
$\Downarrow$ ✓ porten a reescriure la paraula buida

$$\begin{cases} W \to a\underline{X}Y\,ba\,Z\underline{T} \\ W \to aY\,ba\,ZT \\ W \to aXY\,ba\,Z \\ W \to aY\,ba\,Z \end{cases}$$

$\{X \to \alpha_1 \ldots \alpha_n \mid \exists X \to \beta_1 \ldots \beta_n \in G$ i cada α_i compleix que $\alpha_i = \beta_i$ ó (β_i és anul·lable si $\alpha_i = \lambda$) i a més hi ha algun $\alpha_i \neq \lambda.\}$

$L(G) = L(G) - \{\lambda\}$ Es pot demostrar les dues direccions per inducció sobre el nombre de passos de reescriptura.

2ª Transformació: Eliminació de produccions unàries ($X \to Y$) (variable reescriu en variable).

Definició: una variable Y és accessible unàriament des d'X si existeix una derivació $X \to^* Y$ de la forma $X \to X_1 \to X_2 \dots \to X_n = Y$

com calculem totes les accessibles unàriament des d'X per totes les X?

1. Per cada variable X inicialitzem $AU(X) = \{X\}$.
2. Mentre hi hagin X, Y tals que $X \to Y \in G$ i $AU(Y) \not\subseteq AU(X)$, fem $AU(X) = AU(X) \cup AU(Y)$.

Està clar que tot element afegit a $AU(X)$ és arribable des d'X unàriament. La demostració en sentit contrari es faria per inducció sobre el nombre de passos de reescriptura.

= Depurem les següents gramàtiques:

(1) $S \to (S) \mid \lambda$ ⇝ S és anul·lable $S \to (S) \mid ()$

(2) $S \to BC \mid \lambda$
$A \to aA \mid \lambda$
$B \to bB \mid \lambda$
$C \to a$

⇝ anul·lables $= \{S, A, B\}$
$S \to BC \mid C$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to a$

$AU(S) = \{S, C\}$
$AU(A) = \{A\}$
$AU(B) = \{B\}$
$AU(C) = \{C\}$

⇝ $S \to BC \mid a$
$A \to aA \mid a$
$B \to bB \mid b$
$C \to a$

(3) $S \to \textcircled{A} \mid \textcircled{B}$ ⟶ accessibles unàriament des d'S.
$A \to aA \mid aAb \mid a$
$B \to Bb \mid aBb \mid b$

$AU(A) = \{A\}$
$AU(B) = \{B\}$
$AU(S) = \{S, A, B\}$

parts dretes de S que no són variables

⇝ $S \to aA \mid aAb \mid a \mid Bb \mid aBb \mid b$
$A \to aA \mid aAb \mid a$
$B \to Bb \mid aBb \mid b$

* Eliminem les produccions unàries.

Donada G construïm G' sense produccions unàries i tal que $L(G) = L(G')$ amb les regles: $\{X \to w \mid \exists Y \text{ tq } Y \in AU(X) \wedge Y \to w \in G \wedge w$ no és una variable$\}$

Aquesta transformació és polinòmica.

Una gramàtica és quasí-λ-exempta si o bé no té cap λ-producció, o bé només hi és $S \to \lambda$ (símbol inicial), cas en el qual S no apareix a la dreta de cap producció (no hi ha cap regla de la forma $X \to \alpha_1 \dots \alpha_i S \alpha_{i+1} \dots \alpha_n$).

· Transformació a quasí-λ-exempta:

1. Eliminar λ-produccions.
2. En el cas que S fos anul·lable, originàriament, afegim un nou S' com a símbol inicial amb les regles $S' \to S \mid \lambda$.

L'Eliminació posterior de produccions unàries no ens fa perdre el fet de ser quasí-λ-exempta.

* Eliminació de variables no útils.

Una variable és no útil si no existeix una derivació.

$S \to \dots \to \alpha X \beta \to \dots \to \omega$.

paraules qsv sobre terminals i no terminals

paraules terminals

una variable X és no productiva si no existeix una derivació $X \to^* \underset{\text{terminal}}{\underline{\omega}}$

Una variable és no accessible si no existeix una derivació $S \to \dots \to \alpha X \beta$.

Ex(2): $S \to BC \mid \lambda$
$A \to aA \mid \lambda$
$B \to bB \mid \lambda$
$C \to a$

no accessibles: A
no productives: B
no útils: A, B, C

Per eliminar tots els símbols no útils s'han d'eliminar primer els no productius, i després els no accessibles de la gramàtica resultant del pas anterior (l'eliminació d'un símbol implica l'eliminació de totes les regles on aparegui).

EX(2): *Elim. no productius*

$$\begin{array}{l} S\to\lambda \\ \left.\begin{array}{l} A\to aA|\lambda \\ C\to a \end{array}\right\} \rightsquigarrow \text{no accessible} \longrightarrow A, C \rightsquigarrow S\to\lambda \end{array}$$

* càlcul de símbols no útils

29 Abril 05

- càlcul de productius — té una derivació en un símbol terminal
  1. $\overset{\swarrow \text{ var productives}}{P} := \{ X \mid X\to\omega\in G \text{ i } \omega \text{ és terminal}\}$
  2. Mentre hi hagi $X\notin P$ tal que existeix $X\to\alpha_1\ldots\alpha_n\in G$ complint que cada α_i o bé és un terminal o bé $\alpha_i\in P$, llavors fer $P := P\cup\{X\}$.
- càlcul dels accessibles
  1. $A := \{S\}$
  2. Mentre hi hagin $X\notin A$, $Y\in A$ i alguna regla $Y\to\alpha_1\ldots\alpha_i X\alpha_{i+1}\ldots\alpha_n\in G$ fer $A := A\cup\{X\}$.

Per eliminar els símbols no útils de G, s'han d'eliminar primer els no productius i després els no accessibles de la gramàtica resultant de la primera eliminació.

Depuració d'una gramàtica: transformar-la en quasí-λ-exempta, eliminar produccions unàries i finalment eliminar símbols no útils.

Definició: una gramàtica quasí-λ-exempta es diu que està en forma normal de Chomsky si totes les regles (excepte potser $S\to\lambda$) són de la forma $X\to YZ$ ó $X\to a$.

* Transformació a Forma Normal de Chomsky.

1. Depurem la gramàtica i per cada constant c (o símbol terminal) ens inventem una variable X_c, i en cada regla $X\to\alpha$ on $\alpha\geq 2$ reemplacem les ocurrències de cada c per la corresponent X_c, i per cada c afegim la regla $X_c\to c$.
2. Després d'aquest primer pas es preserva el llenguatge generat, i totes les regles (excepte potser $S\to\lambda$) són de la forma $X\to a$ ó $X\to X_1X_2\ldots X_n$ amb $n\geq 2$.

$$\text{EX FNC: } \left\{\begin{array}{l} \boxed{X\to ab} \\ \left.\begin{array}{l} X\to X_aX_b \\ X_a\to a \\ X_b\to b \end{array}\right\} \end{array}\right.$$

3. En el pas següent, per cada regla $X\to X_1X_2\ldots X_n$ amb $n\geq 3$ afegim variables noves $Y_2\ldots Y_{n-1}$, i substituïm la regla anterior per:

$$\begin{array}{l} X\to X_1Y_2 \\ Y_2\to X_2Y_3 \\ Y_3\to X_3Y_4 \\ \quad\vdots \\ Y_{n-1}\to X_{n-1}X_n \end{array}$$

$\swarrow$ si posem Y_n tindríem una producció unària.

Exercici: Transformar a forma Normal de C

(a) $S \to XY$
$X \to aXb \mid \lambda$
$Y \to bYc \mid \lambda$

1 Depurar ≡ Eliminar λ-produccions

(1a). Calculem símbols anul·lables ⇒ Els que transcriuen λ.

Anul·lables $= \{X, Y, S\}$

$S \to XY \mid X \mid Y$ ← Decidim anticipadament anul·lar Y o bé X.
$X \to aXb \mid ab$
$Y \to bYc \mid bc$
$S' \to S \mid \lambda$

Generem el mateix llenguatge tret de la paraula buida que l'afegim anticipadament.

(1b). Eliminem les produccions unàries. → variables accessibles a partir de reescriptura unària.

Augmentem iteratívament els conjunts.

$AU(S') = \{S', S, X, Y\}$ ← arribem des de S.
$AU(S) = \{S, X, Y\}$
$AU(X) = \{X\}$
$AU(Y) = \{Y\}$

↑ tothom arriba a ell mateix

↓

Eliminem produccions unàries → per cada variable posem les parts dretes de les variables que siguin accessibles unàriament

$S' \to \lambda \mid XY \mid aXb \mid ab \mid bYc \mid bc$
$S \to XY \mid aXb \mid ab \mid bYc \mid bc$
$X \to aXb \mid ab$
$Y \to bYc \mid bc$

Càlcul símbols productius ⇝ variables que es transcriuen a paraula terminal. (no productius no ens deixen sortir)

$P := \{X, Y, S, S'\}$

↓ Totes transcriuen a paraula terminal
⇒ no hi ha símbols no productius.

variables accessibles des del símbol inicial S'

$A := \{S', X, Y\}$

Totes les variables de S' són accessibles.
Després cal mirar X,Y però només té com accessibles elles mateixes i ja estan accessibles.
S no és accessible ⇒ A tot arreu on hi hagi S cal eliminar-ho → S s'ha tornat no útil

⇒ Eliminem no accessibles

$S' \to \lambda \mid XY \mid aXb \mid ab \mid bYc \mid bc$
$X \to aXb \mid ab$
$Y \to bYc \mid bc$

Ara transformem a forma normal de Chomsky. ⇒ Afegim les corresponents variables per cada constant.

↙ s'accepta que hi hagi una λ-producció ↘ però només des del símbol inicial.

$S' \to \lambda \mid XY \mid X_aXX_b \mid X_aX_b \mid X_bYX_c \mid X_bX_c$
$X \to X_aXX_b \mid X_aX_b$
$Y \to X_bXX_c \mid X_bX_c$
$X_a \to a$
$X_b \to b$
$X_c \to c$

→ ! Necessitem una variable més → Arreglem $n \geq 3$ $X_a X X_b$ ja que
$X_b X X_c$

FNChomsky són 2var o una constant
$\Rightarrow n \leq 2$

FNC

$S' \to \lambda \mid XY \mid X_a Z_1 \mid X_a X_b \mid X_b Z_2 \mid X_b X_c$
$Z_1 \to X X_b$
$Z_2 \to Y X_c$
$X \to X_a Z_1 \mid X_a X_b$
$Y \to X_b Z_2 \mid X_b X_c$
$X_a \to a$
$X_b \to b$
$X_c \to c$

(b) $S \to SS \mid (S) \mid \lambda$

Depurar :

símbols anul·lables $= \{S\}$

↓

$S' \to S \mid \lambda$
$S \to SS \mid (S) \mid ()$

↓

$AU(S') = \{S', S\}$
$AU(S) = \{S\}$

↓ Eliminem produc unàries

$S' \to SS \mid \lambda \mid (S) \mid ()$

$P := \{S, S'\}$
$A := \{S', S\}$

↓

No tenim símbols no útils a eliminar

↓ FNC – Afegim var per cada constant

$S' \to \lambda \mid SS \mid X_(S X_) \mid X_(X_)$
$S \to SS \mid X_(S X_) \mid X_(X_)$
$X_(\to ($
$X_) \to)$

↓ $n \geq 3$

$S' \to \lambda \mid SS \mid X_(Z \mid X_(X_)$
$S \to SS \mid X_(Z \mid X_(X_)$
$Z \to S X_)$
$X_(\to ($
$X_) \to)$

FNChomsky.

Totes aquestes transformacions preserven la NO ambigüetat: si la gramàtica original era no ambigua, la final també ho és.

Fixada una gramàtica G fixa, considerem el següent problema:

Entrada: paraula terminal ω
Pregunta: és ω generada per G?

Aquest problema és decidible i en temps polinòmic $O(|w|^3)$ l'algorisme assumeix que G està en Forma Normal de Chomsky.

Entrada: taula $\omega[1..n]$ de símbols
Anirem calculant conjunts $t[i,j]$ amb $i \leq j$ de símbols variables que corresponen exactamenta aquelles variables que generen $\omega[i,j]$.

```
des de $i:=1$ fins $n$ fer
  $t[i,i]:=\{X \mid X\rightarrow w[i]\in G\}$
  des de $i:=j-1$ fins $1$ fer
    des de $j':=i$ fins $j-1$ fer

      mentre hi hagin variables $X,Y,Z$ tals que $X\rightarrow YZ\in G$
      $y\in t[i,j']$, $z\in t[j'+1,j]$ i tals que $x\in[i,j]$
      fer
          $t[i,j] = t[i,j]\cup\{X\}$
```

donem sortida cert si $S\in t[1,n]$ i fals en cas contrari.

* Transformació a Forma Normal de Chomsky.

- Gramàtica amb símbol inicial S i amb les següents produccions:

$S \to aSa \mid bSb \mid D \mid AB \mid \lambda$
$A \to aaA \mid aa$
$B \to Bbb \mid b$
$D \to bD \mid Db$
$E \to cDc \mid FS$
$F \to DF \mid cE \mid \lambda$.

1. Depurem. Eliminem les λ-produccions

- Variables anul·lables: $A := \{S, F, E\}$

Gramàtica quasi-λ-exempta

$S' \to S \mid \lambda$
$S \to a\underline{S}a \mid b\underline{S}b \mid D \mid AB \mid aa \mid bb$ $\leftarrow$ $\underline{a}\overset{\lambda}{S}\underline{a}$ $\quad \underline{b}\underset{\lambda}{S}\underline{b}$
$A \to aaA \mid aa$
$B \to Bbb \mid b$
$D \to bD \mid Db$
$E \to cDc \mid FS \mid F \mid S$
$F \to DF \mid cE \mid D \mid c$

2 Eliminem les produccions unàries $\to$ variables accessibles a partir de reescriptura unària iterativament

$AU(S') = \{S', S, D\}$
$AU(S) = \{S, D\}$
$AU(A) = \{A\}$
$AU(B) = \{B\}$
$AU(D) = \{D\}$
$AU(E) = \{E, F, S, D\}$
$AU(F) = \{F, D\}$

Trobem les produccions:

$S' \to \lambda \mid aSa \mid bSb \mid AB \mid aa \mid bb \mid bD \mid Db$
$S \to aSa \mid bSb \mid AB \mid aa \mid bb \mid bD \mid Db$
$A \to aaA \mid aa$
$B \to Bbb \mid b$
$D \to bD \mid Db$
$E \to cDc \mid FS \mid DF \mid cE \mid c \mid aSa \mid bSb \mid AB \mid aa \mid bb \mid bD \mid Db$
$F \to DF \mid cE \mid c \mid bD \mid Db$

3. Eliminació de símbols no útils

- Símbols Productius: variables que transcriuen a símbol terminal

$P := \{S', S, A, B, E, F\}$

- Símbols No productius $\to$ D

* Eliminem no productius:

$S' \to \lambda \mid aSa \mid bSb \mid AB \mid aa \mid bb$
$S \to aSa \mid bSb \mid AB \mid aa \mid bb$
$A \to aaA \mid aa$
$B \to Bbb \mid b$
$E \to FS \mid cE \mid c \mid aSa \mid bSb \mid AB \mid aa \mid bb$
$F \to cE \mid c$

- símbols no accessibles des del símbol inicial S'.

$NAc := \{E, F\}$

* Eliminem NO accessibles:

$S' \to \lambda \mid aSa \mid bSb \mid AB \mid aa \mid bb$
$S \to aSa \mid bSb \mid AB \mid aa \mid bb$
$A \to aaA \mid aa$
$B \to Bbb \mid b$

1. Forma Normal de Chomsky.

Afegim les corresponents variables per cada constant.

$S' \to \lambda \mid X_a S X_a \mid X_a X_a \mid X_b S X_b \mid X_b X_b \mid AB$

$S \to X_a S X_a \mid X_a X_a \mid X_b S X_b \mid X_b X_b \mid AB$

$A \to X_a X_a A \mid X_a X_a$

$B \to B X_b X_b \mid X_b$

$X_a \to a$

$X_b \to b$

$\rightsquigarrow$ $n \geq 3 \to$ Han de ser de la forma $X \to X_1 X_2$

o bé

$X \to a$

FNC

$S' \to \lambda \mid X_a Z_1 \mid X_a X_a \mid X_b Z_2 \mid X_b X_b \mid AB$

$Z_1 \to S X_a$

$Z_2 \to S X_b$

$S \to X_a Z_1 \mid X_a X_a \mid X_b Z_2 \mid X_b X_b \mid AB$

$A \to X_a Z_3 \mid X_a X_a$

$B \to B Z_4 \mid X_b$

$Z_3 \to X_a A$

$Z_4 \to X_b X_b$

$X_a \to a$

$X_b \to b$

vàrem veure que era decidible saber donades gramàtiques (G) i paraules (ω), si G genera ω.

A més, considerant només ω com a entrada (G fix i tenim una família de problemes, un per cada G) era resoluble amb cost $O(|\omega|^3)$.
També es pot aconseguir cost polinòmic tenint la G d'entrada, però s'hauria de refinar l'algorisme que vàrem veure (eliminar λ-prod's té cost exponencial)
També és decidible si donada G, genera $\overline{\Phi}$, doncs és equivalent a veure que S és un símbol no productiu.

Problemes indecidibles amb Gramàtiques:

1. Entrada: G1, G2.
   Problema: saber si G1 i G2 poden generar una mateixa paraula
2. Entrada: G
   És G ambigua

Per justificar-ho, reduïrem des de PCP.

Entrada:

$P = \{(u_1, v_1), \_\_, (u_n, v_n)\}$

Sortida:

1. G1, G2 amb paraula comuna si i només si P té solució.
2. G ambigua si i només si P té solució.

1. Reducció.

Puc escollir $(u_{i1}, v_{i1}), \dots, (u_{ik}, v_{ik})$ tal que $u_{i1}u_{i2} \dots u_{ik} = v_{i1}v_{i2} \dots v_{ik}$

$G1: S_1 \to u_1 S_1 p_1 \mid u_2 S_2 p_2 \mid \dots \mid u_n S_1 p_n \mid u_1 p_1 \mid u_2 p_2 \mid \dots \mid u_n p_n$

$G2: S_2 \to v_1 S_2 p_1 \mid v_2 S_2 p_2 \mid \dots \mid v_n S_2 p_n \mid v_1 p_1 \mid \dots \mid v_n p_n$

Afegim n símbols terminals nous $\{p_1, \dots, p_n\}$

2. Reducció.

$S \to S_1 \mid S_2$ on S_1 i S_2 tenen les regles anteriors

- Altres problemes indecidibles

  saber si una gramàtica genera Σ^*.
  saber si dues gramàtiques generen el mateix llenguatge.

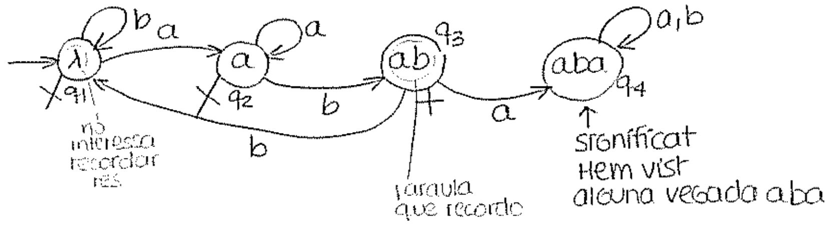
- <u>Autòmats i expressions regulars</u>

<u>Autòmats finits</u>: Només tenen estat i transicions, on aquestes estan anotades amb símbols, que són la condició de lectura de l'entrada per efectuar la transició (≡ canvi d'estat de la màquina).

<u>Exemple</u>:

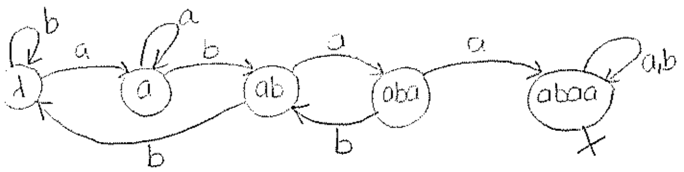
- llenguatge sobre $\{a, b\}$ que no contenen aba

$\{\omega \in \{a,b\}^* \mid \neg \exists xy : \omega = xabay\}$



aabbaba
$q_1 q_2 q_2 q_3 q_1 q_2 q_3$

- Accepta el patró p = abaa sigui $\Sigma = \{a,b\}$

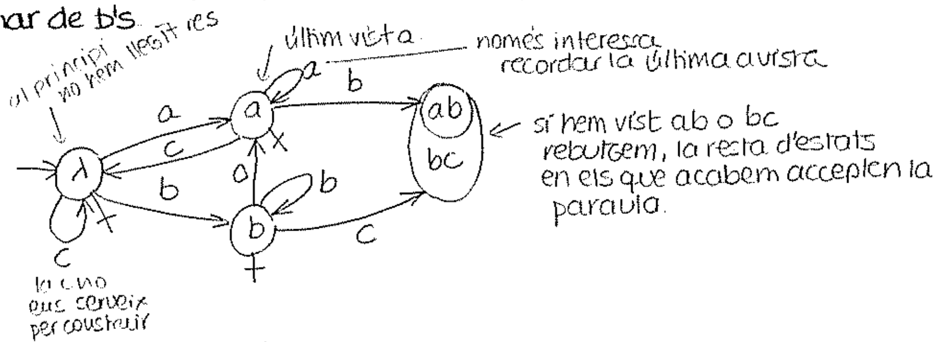


si m = |p| llavors existeix un DFA amb m+1 estats

Exercici : Trobeu autòmats per:

(a) Paraules sobre {a,b,c} que no continguí ni la subparaula ab ni bc
(b) Paraules sobre {a,b,c} que no continguín ni la subparaula ab i bc al mateix temps.
(c) Paraules sobre {a,b} que contenen aba però no bab
(d) Paraules sobre {a,b} amb un nombre parell de a's i un nombre senar de b's.

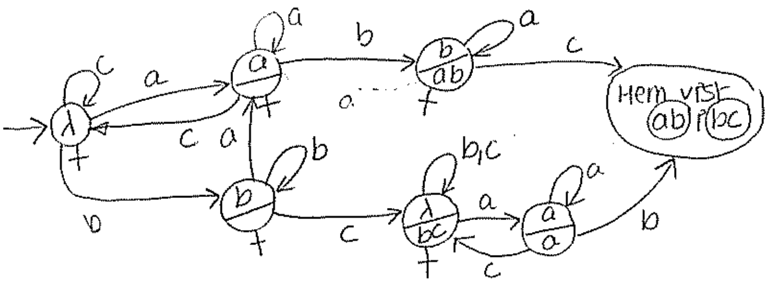
(a)



si hem vist ab o bc rebutgem, la resta d'estats en els que acabem acceptem la paraula.

Rebutgem paraules ab ...
bc ...
ó si conté ab ... bc
bc ... ab
abc

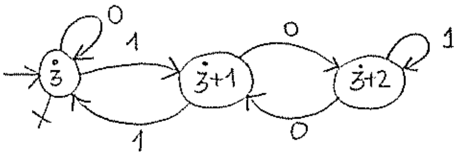
(b)



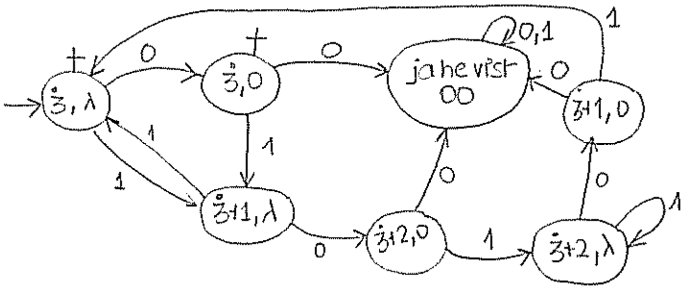
2 Maig 05

∘ $\{w \in \{0,1\}^* \mid valor(w) = \dot{3}\}$

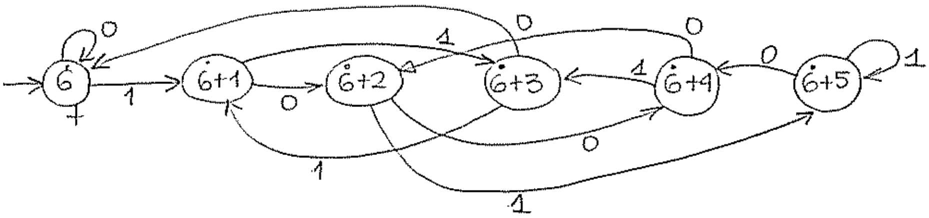
si $\dot{3}$ veu un 0 es multiplica ×2 ⇒ $\dot{3}$
si $\dot{3}$ veu un 1 es multiplica ×2 i es suma 1 ⇒ $\dot{3}+1$



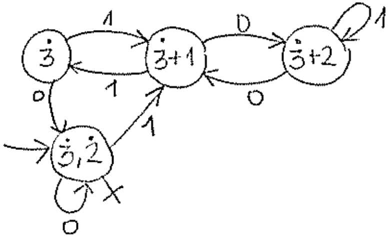
- Paraules sobre $\{0,1\}$ que representen $\dot{3}$ i que no tenen 2 zeros seguits.



- Paraules sobre $\{0,1\}$ que representen $\dot{3}$ i $\dot{2}$

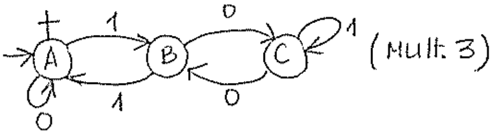


més simple:

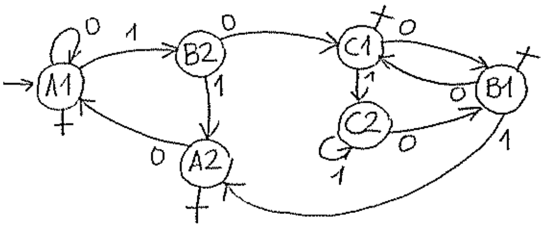


<u>Def</u>. Els llenguatges regulars són reconeixibles per un DFA (Autòmat Finit Determinista).
Els llenguatges regulars són tancats per complementari, intersecció i unió.

Aquí, determinisme ≡ totes les transicions estan definides, i unívocament.
Llavors, complementari ≡ intercanviar estats acceptadors per <u>no</u> acceptadors.
intersecció ≡ fer un nou autòmat que simuli als dos donats, i posar com acceptadors els estats que ho siguin dels 2 donats.
unió ≡ feu un nou autòmat que simuli els dos donats, i posar com acceptadors els estats que ho siguin d'algun dels dos donats.



Simulem els 2 autòmats alhora.



Acceptadors → tots els que tinguin A o 1.

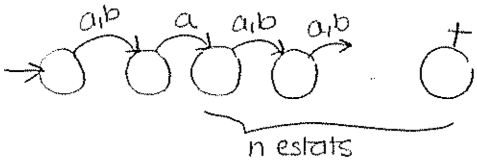
(Autòmat finit indeterminista)

- **NFA** : Conjunt d'autòmats no deterministes, ja que accepten transicions no definides, transicions multiplement definides, i varis estats inicials. Diem que un NFA A accepta una paraula w si existeix un camí d'execució d'A amb entrada w que acaba en estat acceptador.

Exemple (suposem un n fixat)

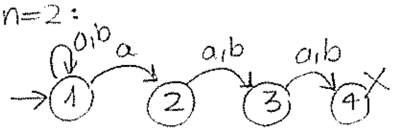
$$\{w\in\{a,b\}^* \mid w\in\{a,b\}^* a\{a,b\}^n\} = \{w\in\{a,b\}^+ \mid \exists x,y : w=xay \wedge |y|=n\}$$

Aquest llenguatge es pot reconeixer amb un NFA de $n+2$ estats. $L_n=\Sigma^* a\Sigma^n$



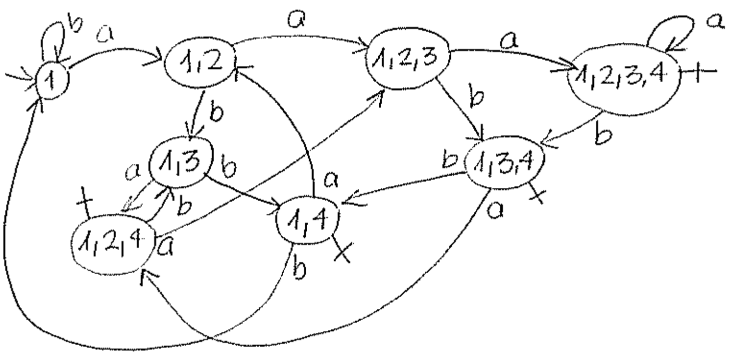
El DFA més petit reconeixent el mateix llenguatge té 2^{n+1} estats.

Per $n=2$:



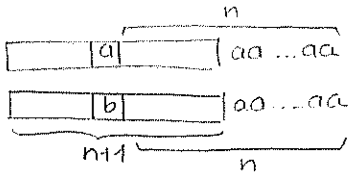
× Transformació de NFA a DFA preservant el llenguatge. ⇒ Determinització

Acceptadors: tots els que contineu el 4.



- verificació de la cota inferior 2^{n+1} per DFA's reconeixent $\{a,b\}^* a\{a,b\}^n$.

Es demostra per reducció al absurd. suposem que un DFA A amb menys que 2^{n+1} estats reconeix $\{a,b\}^* a\{a,b\}^n$.



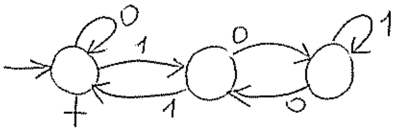
De paraula de mida $n+1$ n'hi ha 2^{n+1}, i com que hi ha més paraules que estats, n'hi ha dues de diferents que ens porten al mateix estat q.

Com que x,y són diferents, necessàriament són de la forma $x=x_1ax_2$, $y=y_1by_2$ on $|x_1|=|y_1|$, $|x_2|=|y_2|<n+1$, les paraules $xa^{n-|x_2|}$ i $ya^{n-|y_2|}$ també ens porten al mateix estat, però una és del llenguatge i l'altra no ⇒ contradicció ja que una s'hauria d'acceptar i l'altra no.

9 Maig 05

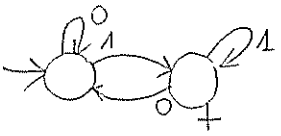
Els llenguatges regulars són tancats per l'operació de revessat: L regular $\Rightarrow$ L^R regular. Donat un cert autòmat, se'n pot construir un altre que reconegui el revessat de l'anterior, a base de, simplement, canviar la direcció de les fletxes (arestes o transicions) i posar els estats finals com a inicial, i l'inicial com a final.

— Exemple: múlt de 3

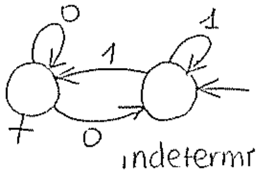


$\overset{R}{\Longrightarrow}$ queda igual curiosament.

• múlt. de 2 + 1

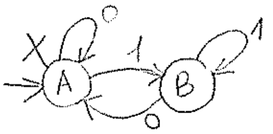


$\overset{R}{\Longrightarrow}$

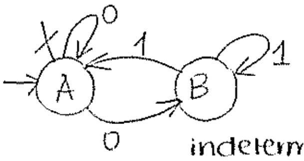


indeterminista!

• múlt. de 2

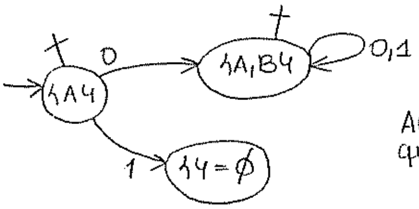


$\overset{R}{\Longrightarrow}$



indeterminista!

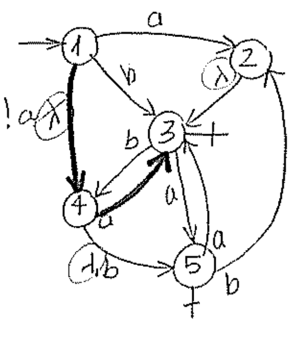
= Determinitzem el revessat de $\dot{2}$. $\{\omega \in \{0,1\}^* \mid \text{valor}(\omega^R) = \dot{2}\}$



Autòmat determinista que accepta el revessat de $\dot{2}$.

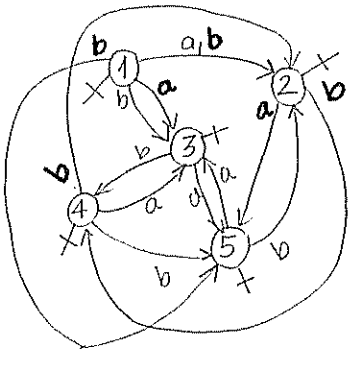
Els llenguatges regulars també són tancats per concatenació. Per veure-ho, variem el model i admeterem λ-transicions. En aquest cas, λ-NFA's, i.e., autòmats indeterministes amb λ-transicions. Una λ-transició es pot executar sempre, i sense llegir l'entrada. La noció d'acceptació és la mateixa que per als NFA's, i s'accepta si existeix un camí d'execució acceptador.

Exemple: "M'acabo d'inventar això que no sé què reconeix" (Guillem Godoy)



paraules reconegudes: a, b, λ ... (segurament) $\{a,b\}^*$

(! Ara eliminarem les λ-transicions, però fixem-nos que d'1 podem arribar a 3 amb una a, gràcies a la λ-transició. Així que cal afegir la transició λ $\Rightarrow$ ① $\xrightarrow{a}$ ③ .... ① $\xrightarrow{b}$ ⑤ ....)



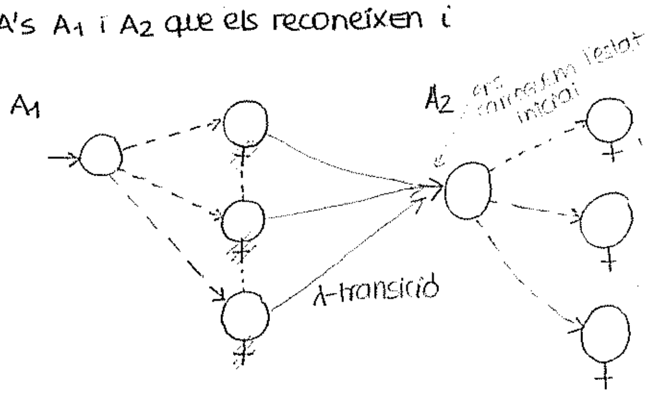
En general, per eliminar λ-transicions:

1. si desde q_1 s'arriba a q_2 via λ-transició, i tenim una $\delta(q_2,a)=q_3$ llavors hem d'afegir $\delta(q_1,a)=q_3$. (seria més correcte dir que afegim q_3 a $\delta(q_1,a)$ ja que podem tenir indeterminisme).
2. si desde q_1 arribem a estat acceptador via λ-trans, llavors hem de posar q_1 com a acceptador.

Els llenguatges regulars són tancats per concatenació.

Donats L_1, L_2 regulars (llenguatges reconeguts per un autòmat finit i determinista), tenen DFA's A_1 i A_2 que els reconeixen i

$\Rightarrow$)



Estats acceptadors de la concatenació.

Autòmat indeterminista degut a les λ-transicions.

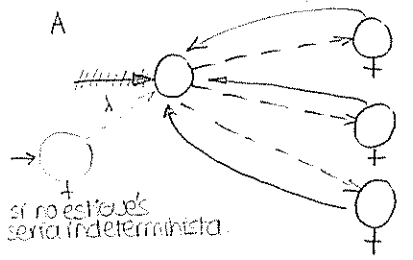
$\Leftarrow$) Qualsevol paraula que pertanyi a L_1L_2 sempre existeix una partició (subparaula) que accedeix des de l'inici del autòmat fins a un dels antics estats acceptadors de L_1 i l'altra subparaula (després d'executar l'λ-transicions) accedeix fins l'estat acceptador de L_2.

Per construir l'autòmat que reconeix L_1L_2 n'hi ha prou amb:

- Afegir λ-transicions des dels estats finals de A_1 cap l'inicial d'A_2.
- Posar com a estats finals només els que eren finals d'A_2.
- Posar com a estat inicial el que era inicial a A_1

Els llenguatges regulars són tancats per l'estrella de Kleene. Donat L, llenguatge regular, amb DFA A, podem construir DFA per

$L^* = \underbrace{L^0}_{\text{paraula buida}} \cup L^1 \cup L^2 \ldots$



cal contemplar la paraula buida

a base de:

- Afegir λ-transicions des dels estats acceptadors cap a l'estat inicial.
- Afegir un nou estat que serà l'inicial i acceptador també, i amb una λ-transició, cap a l'antic estat inicial d'A.

Els llenguatges regulars també són tancats per L^+ (Tancament positiu) (Pot contenir λ si $\lambda \in L$, en canvi L^* sempre conté la paraula buida). $L^+ = L^1 \cup L^2 \cup L^3 \ldots$ i es justifica amb la primera transformació: $\boxed{\text{si } \lambda \in L \Rightarrow L^+ = L^*}$

Exemple: construcció de l'autòmat fent servir operacions:

Paraules sobre $\{a,b\}$ tals que a la dreta de cada a hi ha un nombre parell de b's.

Formalització: $\{w \in \{a,b\}^* \mid \forall x,y\ (w = xay \Rightarrow |y|_b = \dot{2})\}$

Procedim a trobar un autòmat que el reconegui:

Passem al complementari; perquè és més fàcil trobar-ne l'autòmat → Els existeix tenen autòmats indeterministes.

$$\overline{L} = \{w \in \{a,b\}^* \mid \exists x,y : (w = xay \wedge |y|_b \neq \dot{2})\}$$

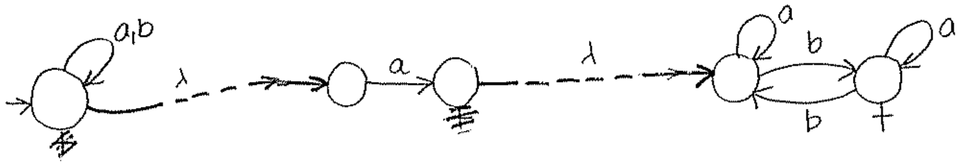
$\boxed{\neg(\forall x\, (R(x) \Rightarrow S(x)) \Leftrightarrow (\exists x\, (R(x) \wedge \neg S(x)))}$

Descomposem la definició de $\overline{L}$ en:

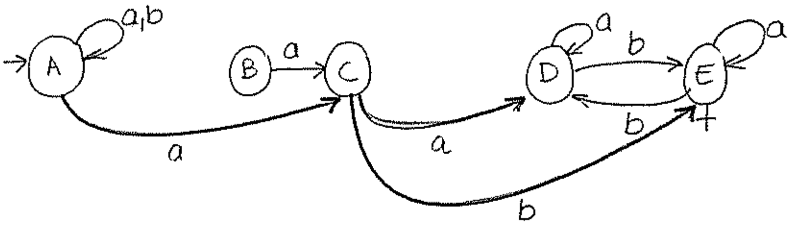
$$\overline{L} = \{x \in \{a,b\}^*\}\{a\}\{y \in \{a,b\}^* \mid |y|_b \neq \dot{2}\}$$

Concatenem els llenguatges :

$\{a,b\}^*$ $\{a\}$ $\{y \in \{a,b\}^* \mid |y|_b \neq \dot{2}\}$

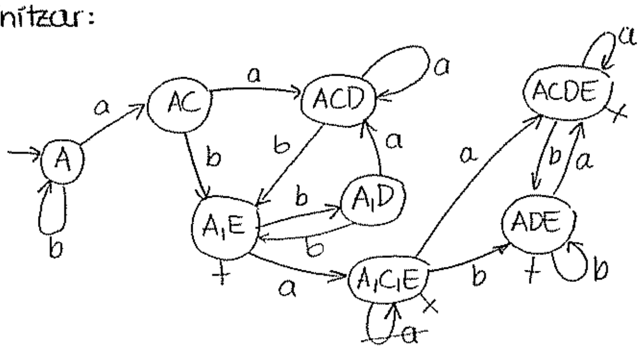


Per Eliminar λ-transicions i preservar el llenguatge el que fem és anticipar-nos ⟹



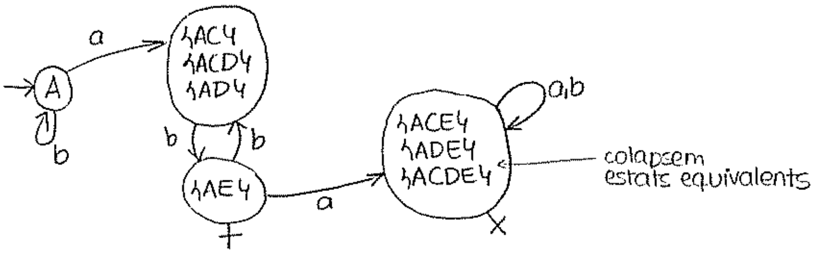
És indeterminista

Determinitzar:

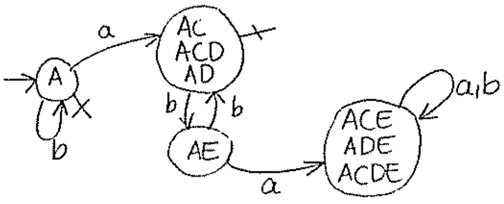


Posem com a acceptadors tots els estats que tinguin una E

Minimitzar l'autòmat :



Passem al complementari : (Posant com a estats acceptadors aquells que no ho són)



== Exercici 2: Trobar un autòmat per:

Paraules sobre {a,b} tals que tot prefix que acaba en a, conté un nombre parell de b's i tot sufix que comença per b conté un nombre parell d'a's

Formalització : $L = \{w \in \{a,b\}^* \mid (\forall xy: w = xay \Rightarrow |x|_b = \dot{2}) \wedge (\forall xy: w = xby \Rightarrow |y|_a = \dot{2})\}$

Complementari: $\overline{L} = \{w \in \{a,b\}^* \mid (\exists xy: w = xay \Rightarrow |x|_b \neq \dot{2}) \vee (\exists xy: w = xby \Rightarrow |y|_a \neq \dot{2})\}$

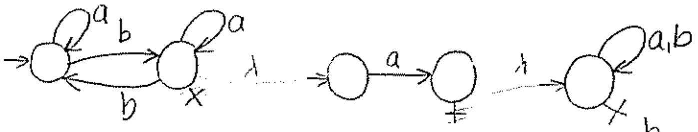
Descompasar:

1. $\{x \in \{a,b\}^* \mid |x|_b \neq \dot{2}\} \cdot \{a\} \{y \in \{a,b\}^*\}$
2. $\{x \in \{a,b\}^*\} \cdot \{b\} \cdot \{y \in \{a,b\}^* \mid |y|_a \neq \dot{2}\}$

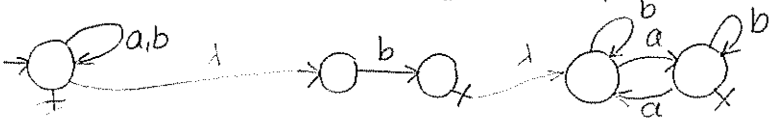
} 1∪2

Autòmats :

1.

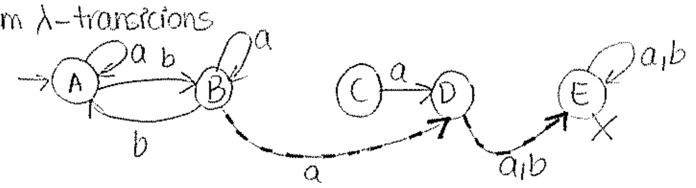


2.

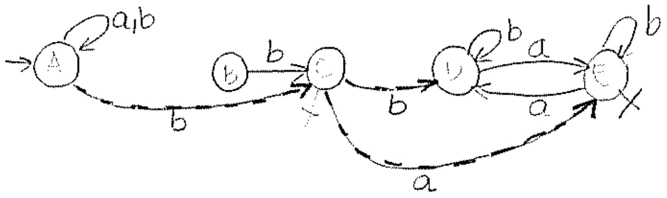


Eliminem λ-transicions

1.

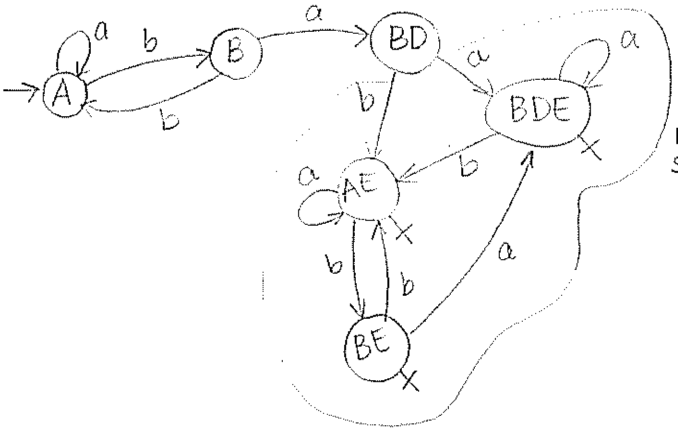


2.



Determinitzar:

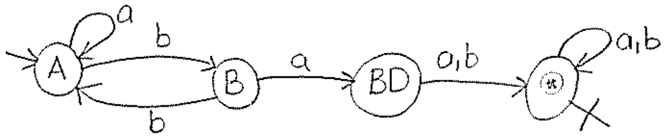
1.



Tots 3 accepten i no es pot sortir d'ells.(*)

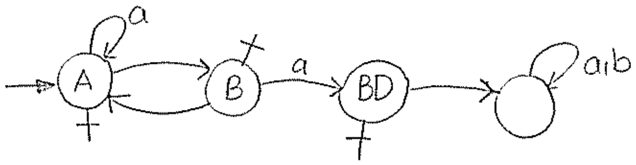
Minimitzar:

1.



Complementar

1.

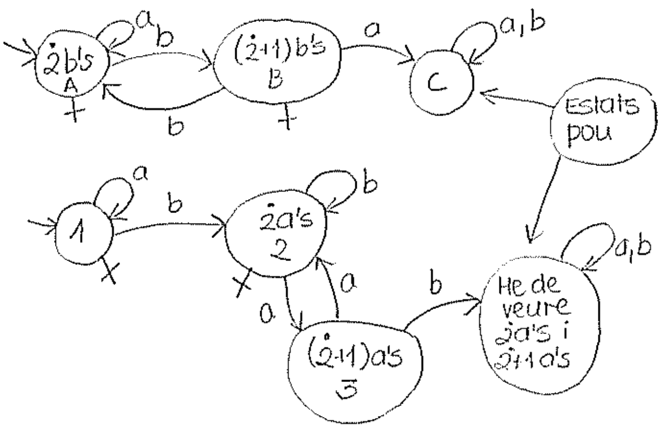


¡ solució professor:

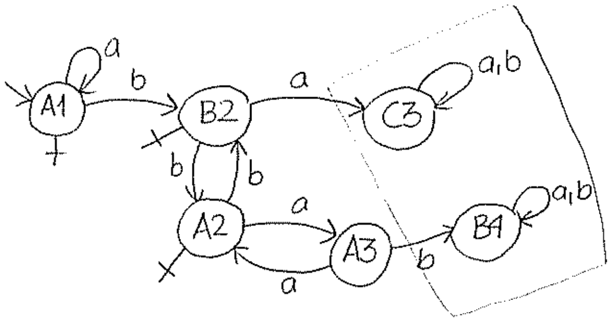
$$L = \underbrace{\{ \omega \mid \forall x,y : (\omega = xay \Rightarrow |x|_b = \dot{2}}_{L1} \} \cap \{ \underbrace{\omega \mid \forall x,y : (\omega = xby \Rightarrow |y|_a = \dot{2}}_{L2} \}$$

A1 :

A2 :



intersecció A_1 i A_2 :



Els estats pou
es poden col·lapsar

Minimitzem ⟹

0-indistingible : {A1, B2 A2} {C3, A3, B4}
1-indistingible : {A1} {B2, A2} {C3, B4} {A3}
2- indistingible : {A1} {B2} {A2} {C3, B4} {A3}
3- indistingible : {A1} {B2} {A2} {C3, B4} {A3} ⟹ Collapsem C3 i B4 i ja tenim l'autòmat mínim.

	a	b
A1	A1	B2
B2	C3	A2
A2	A3	B2
C3	C3	C3
A3	A2	B4
B4	B4	B4

per els conjunts inicials
mirem si amb {|w|= 1} → {A1, B1, A2} amb b
arribem a B2, A2 i
aquest son un conjunt
{B2, A2} 0-ind ⟹ ara 1-ind

	I			II		
	A1	B2	A2	C3	A3	B4
a	I	II	II	II	I	II
b	I	I	I	II	II	II

nova
classe
equivalència

	I	II		III		IV
	A1	B2	A2	C3	B4	A3
a	I	III	IV	III	III	II
b	II	II	II	III	III	III

	I	II	III	IV		V
	A1	B2	A2	C3	B4	A3
a	I	IV	V	IV	IV	III
b	II	III	II	IV	IV	IV

15 Maig05

Minimització d'autòmats finits determinats:

Algunes qüestions de notació: un DFA es sol denotar amb una tupla $<\Sigma, Q, \delta, q_0, F>$ on

Σ : alfabet d'entrada
Q : conjunt d'estats
q_0: estat inicial $\in Q$
$F \subseteq Q$: conjunt d'estats acceptadors
δ : $Q \times \Sigma \rightarrow Q$ funció de transició.

En el cas de NFA's, $\delta \subseteq Q \times \Sigma \times Q$ (fixat un estat de sortida Q i Σ, puc anar a parar a diversos estats $\Rightarrow$ indet). A vegades $\delta(q,a)$ es denota de forma simplificada com $q \cdot a$.
La δ s'extén de forma natural de símbols a paraules: $q \cdot (a_1 a_2 \ldots a_k) = (((q a_1) a_2) a_3 \ldots) a_k)$

El llenguatge reconegut per A és: $\{w \in \Sigma^* \mid q_0 w \in F\}$. També podem parlar de paraules acceptades desde un $q \in Q$, que és $Lq = \{w \in \Sigma^* \mid q \cdot w \in F\}$

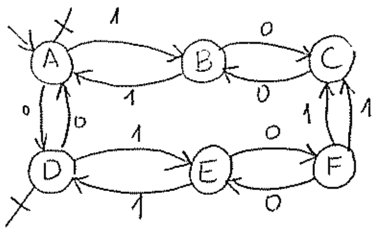
Fixat un DFA $A = <\Sigma, Q, \delta, q_0, F>$. Donats q_1, q_2, diem que són <u>indistingibles</u> si $Lq_1 = Lq_2$, és a dir, per tota $w \in \Sigma^*$: $q_n \cdot w \in F \Leftrightarrow q_2 \cdot w \in F$.

Diem que q_1, q_2 són <u>K-indistingibles</u> si $\forall w \in \Sigma^*$ tal que $|w| \leq K$ es compleix: $q_1 w \in F \Leftrightarrow q_2 w \in F$.
EXEMPLE: dir que q_1 i q_2 són 0-indistingibles equival a dir que q_1, q_2 pertanyen tots 2 a F o $\neg F$.

Definició alternativa (recursiva):

- q_1, q_2 són 0-indistingibles si o bé $q_1, q_2 \in F$ ó bé $q_1, q_2 \notin F$ $\Rightarrow$ separem estats acceptadors dels que no ho són.
- per $K \geq 1$, $q_1 q_2$ són K-indistingibles si són (K-1)-indistingibles, i a més $\forall a \in \Sigma$ $(q_1 a)(q_2 a)$ són (K-1)-indistingibles.
- Cap parella que no es decideixi com a indistingible de les anteriors regles és indistingible.

EXEMPLE:



Partició en 0-indistingibles:
$\{A, D\}$ $\{B, C, E, F\}$

separem els estats acceptadors dels que no ho són.

Partició en 1-indistingibles:
$\{A, D\}$ $\{B, E\}$ $\{C, F\}$

Partició en 2-indistingibles:
$\{A, D\}$ $\{B, E\}$ $\{C, F\}$

tot i que són 2-indp miro un sol mov $|w| = 1$ i mirem que estiguin al mateix conjunt a 1-ind i així fins al infinit.

A i D són 0-ind $\rightarrow$ candidats a ser 1-ind.
A i D amb $|w| = 1$:
Amb 0 van a parar a D, A que estan en el mateix conjunt
amb 1 van a parar a B, E que estan al conj 0-ind per tant són 1-ind

En el moment que la partició en k-indistingibles coincideix amb la de (K-1)-indistingibles, podem aturar el càlcul, ja que llavors sabem que tenim la partició definitiva en indist.

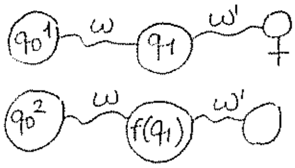
<u>L'autòmat mínim és únic</u>. Qualsevol mínim ha de complir:

- Tots els estats són accessibles des del estat inicial. (En cas contrari podríem esborrar els no accessibles obtenint un DFA menor $\Rightarrow$ contradicció).
- Tota parella d'estats ha de ser distingible; és a dir, tot q_1, q_2 compleix que $Lq_1 \neq Lq_2$ (en cas contrari l'algorisme de minimització produeix un autòmat menor $\Rightarrow$ contradicció).

Suposem que tenim dos autòmats mínims A_1 i A_2 que reconeixen el mateix llenguatge. Construïm una funció $f: Q_1 \rightarrow Q_2$ ($Q_1 \in A_1$ i $Q_2 \in A_2$) d'aquesta manera $\forall q \in Q_1$, és accessible, i per tant, existeix ω tal que $q_0^1 \cdot \omega = q_1$ (q_0^1 és estat inicial de A_1). Definim $f(q_1) = q_0^2 \omega$. Es compleix que:

$L_{q_1} = L_{f(q_1)}$ perquè? : per cada ω' $(\underbrace{q_0^1 \omega}\omega' \in F_1 \Leftrightarrow \underbrace{q_0^2 \omega}\omega' \in F_2)$

$$\begin{gathered} \| \\ (q_1 \omega' \in F_1 \Leftrightarrow f(q_1)\omega' \in F_2) \end{gathered}$$



Si ω' porta a estat acceptador i a baix no, A_1 i A_2 no reconeixen el mateix llenguatge.

Llavors f és injectiva, perquè tots els estats de l'autòmat són distingibles. Faltaria per veure que des de $a = f(q_1) \cdot a$ ($\Rightarrow A_1$ i A_2 són el mateix autòmat)

17 Maig 05

Expressions regulars:

Són expressions (molt simples) que representen llenguatges definibles utilitzant només operacions d'estrella, concatenació i unió sobre elements bàsics $\emptyset, \{\lambda\}, \{a\}$

És típic quan es parla d'expressions regulars representar la unió amb el símbol de la suma (+), i també representar $\{a\}, \{\lambda\}$ directament amb a, Λ (lambda(λ) majús.)

Exemple de representació amb expressions regulars:

- Paraules acabades en a: $(a+b)^* a$
- Paraules amb un nombre parell d'a's: $(b^* a b^* a)^* \boxed{b^*}$ – per a acceptar les que tenen 0 a's.

Els llenguatges representats per expressions regulars són llenguatges regulars (reconeixibles amb un DFA), perquè la classe de llenguatges regulars conté $\emptyset$, $\{\lambda\}, \{a\}$ per tot $a \in \Sigma$ i a més, és tancada per concatenació, estrella i unió. A més es pot construir automàticament el corresponent DFA a una expressió regular donada.

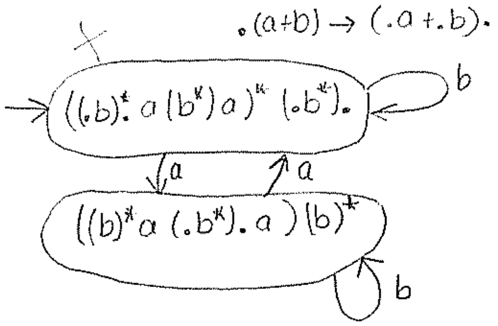
$\Rightarrow$ Construïm directament un autòmat per $(b^* a b^* a) b^*$.

Metodologia: '$\cdot$' significa on estem

$\cdot(b^*) \xrightarrow{b}$ com que és estrella, o llegim el que hi hagi o no (λ)

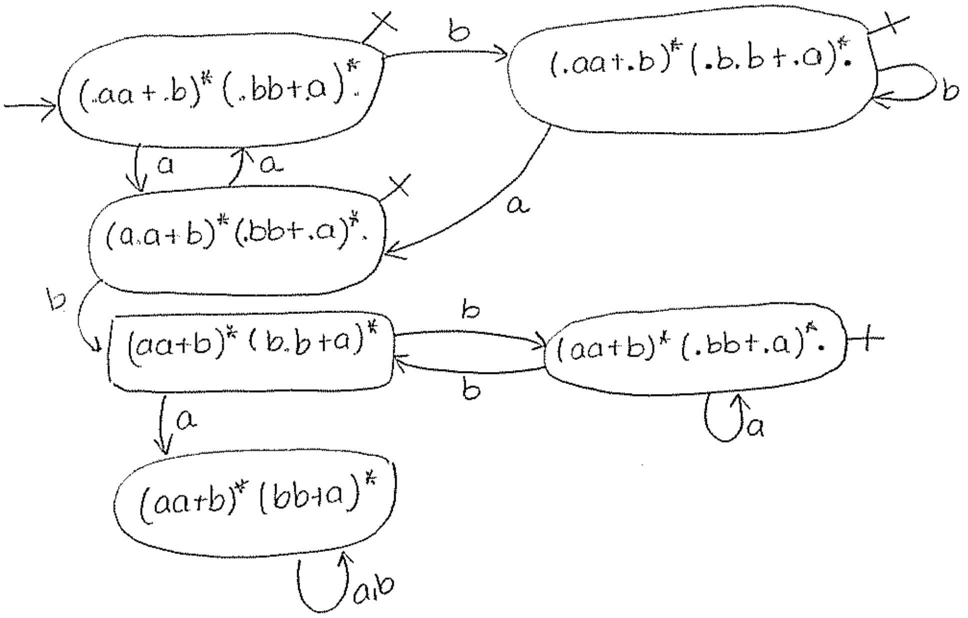
$(\cdot b^*)\cdot$

$\cdot(a+b) \rightarrow (\cdot a + \cdot b)\cdot$

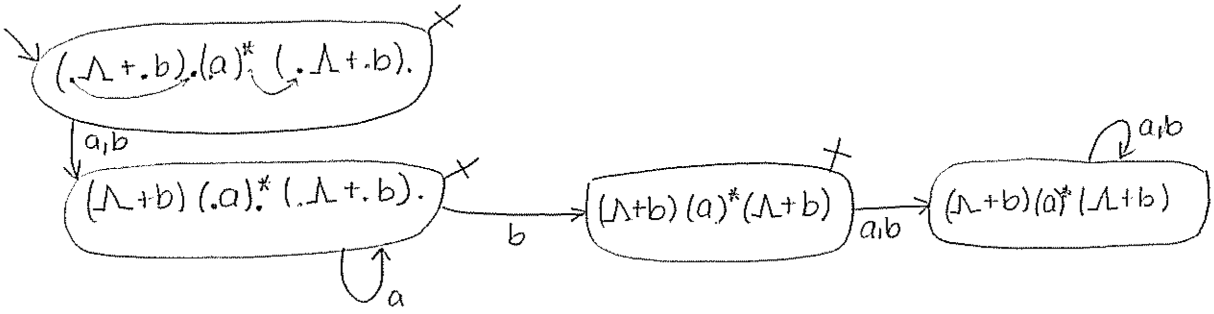


17 Maig 05

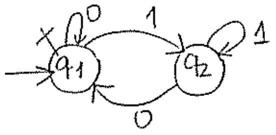
- $(aa+b)^*(bb+a)^*$



- $(\Lambda+b)a^*(\Lambda+b)$



Tot llenguatge regular (és a dir, reconeixible amb un DFA) és representable amb una expressió regular. Partim d'un exemple, per veure que requerirem resoldre un cert tipus d'equacions sobre llenguatges. Paraula sobre {0,1} que representen un $\dot{2}$.



L'objectiu és construir una expressió regular representant el mateix llenguatge.

L_1 = paraules acceptades per l'autòmat si es comença. l'execució des de q_1.

L_2 = paraules acceptades començant l'execució des de q_2.

Condicions necessàries que han de complir L_1 i L_2.

$$\left.\begin{array}{l} L_1 = \Lambda + \emptyset L_1 + 1L_2 \\ L_2 = \emptyset L_1 + 1L_2 \end{array}\right\} \underset{\text{equació}}{\Rightarrow} \quad \left.\begin{array}{l} X_1 = \Lambda + \emptyset X_1 + 1X_2 \\ X_2 = \emptyset X_1 + 1X_2 \end{array}\right\}$$

Aquest sistema té solució única i es pot construir automàticament com a expressió regular.

Per a solucionar aquest sistema utilitzarem el següent lema:

Lema d'Arden: Donats dos llenguatges A i B, i una equació $X = AX + B$,① el llenguatge A^*B és una solució,② qualsevol funció conté el llenguatge A^*B,③ si A no conté λ llavors A^*B és l'única solució.

Veiem:

$$A^*B = \underbrace{A(A^*B)+B}_{(A\cdot A^* + \Lambda)B}$$

Hauríem de justificar abans propietats com la distributiva

$C\cdot D + E\cdot D = (C+E)D$

18 maig 05

sigui L una solució, llavors compleix $L = AL + B$ en particular:

$\longrightarrow B \subseteq L \Rightarrow AB \subseteq AL \subseteq L \Rightarrow AB \subseteq L \Rightarrow AAB \subseteq AL \Rightarrow \dots \Rightarrow A^*B \subseteq L$

Ho veiem per reducció a l'absurd: suposem que $\lambda \notin A$ però que existeix una solució $L \supseteq A^*B$ i $L \neq A^*B$.

Com que L és solució, compleix $L = AL+B$, però a més existeix ω tal que $\omega \in L$ i $\omega \notin A^*B$. Agafem una paraula de les de menor mida ($|\omega|$) d'entre les que estan a L i no a A^*B. Com que $\omega \in A^*B$, $\omega \notin B$, i donat que $\omega \in L = AL+B$, llavors $\omega \in AL$ i per tant és de la forma $\omega = \omega_1\omega_2$ amb $\omega_1 \in A$, $\omega_2 \in L$ on ω_2 no pot ser de B, ja que si no fos $\omega_1\omega_2$ també seria d'A^*B.
Donat que $\lambda \in A$, $|\omega_1| \geq 1$, i per tant $|\omega_2| < \omega \Rightarrow \omega \in L, \omega_2 \in A^*B$ i $|\omega_2| < \omega \Rightarrow$ ¡contradicció amb el fet que ω fos paraula de menor mida complint certes condicions.

→ continuació exemple $\dot{2}$

$$\left.\begin{aligned} L_1 &= \Lambda + 0L_1 + 1L_2 \\ L_2 &= 0L_1 + 1L_2 \end{aligned}\right\}$$

$$L_2 = \underbrace{1L_2}_{A} + \underbrace{0L_1}_{B}$$

$|||$

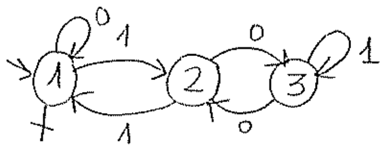
$$X = AX + B \Rightarrow X = A^*B$$

$$L_1 = 0L_1 + 11^*0L_1 + \Lambda \Rightarrow L_1 = \underbrace{(0 + 11^*0)L_1}_{A} + \underbrace{\Lambda}_{B}$$

$$L_1 = A^*B = \underbrace{(0 + 11^*0)^*\Lambda}_{\text{Expressió regular que representa } \dot{2}}$$

L_1 ↓ paraula reconeguda des de L_1 → que és l'estat inicial ⇒ el que ens informa

→ Trobar una expressió regular per $\dot{3}$



$$L_1 = \Lambda + 0L_1 + 1L_2 \Rightarrow L_1 = \Lambda + 0L_1 + 1((01^*0)^*1L_1) = \Lambda + (0 + 1(01^*0)^*1)L_1 =$$

$$= (0 + 1(01^*0)^*1)^*\Lambda$$

$$L_2 = 1L_1 + 0L_3 \Rightarrow L_2 = \underset{B}{\underline{1L_1}} + \underset{A}{\underline{01^*0L_2}} = (01^*0^*)1L_1$$

$$L_3 = \underbrace{1L_3}_{A} + \underbrace{0L_2}_{B} \Rightarrow L_3 = A^*B = 1^*0L_2$$

substituïm.

19-5-05

Donada l'equació $X = AX + B$ sobre llenguatges (A i B són llenguatges fixats, les solucions X són llenguatges, i $+$ s'interpreta com unió):

1 - A^*B n'és una solució

2 - qualsevol solució L compleix $L \geq A^*B$

3 - si $\lambda \notin A$ llavors A^*B n'és l'única solució.

A^*B és llenguatge. $$A^*B = \underbrace{A(A^*}_{\text{1ó+ paraules d'A seguides d'una de B}}B) + B$$

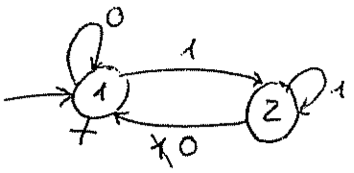
3) Ho veiem per reducció a l'absurd: suposem que $\lambda \notin A$ però que existeix una solució $L \underset{\neq}{\geq} A^*B$ i $L \neq A^*B$

Com que L és solució, compleix $L = AL+B$, però a més existeix w tal que $w \in L$ i $w \notin A^*B$. Agafem una paraula w de les de menor mida ($|w|$) d'entre les que estan a L i no a A^*B.

Com que $w \notin A^*B$, $w \notin B$, i donat que $w \in L = LA + B$, llavors $w \in AL$ i per tant és de la forma $w = w_1 w_2$ amb $w_1 \in A$ i $w_2 \in L$. w_2 no pot ser de A^*B, doncs si ho fós $w_1 w_2$ també seria de A^*B. Donat que $\lambda \notin A$, $|w_1| \geq 1$, i per tant $|w_2| < w$.

Resumint: $w_2 \in L$, $w_2 \notin A^*B$ i $|w_2| < |w|$. CONTRADICCIÓ amb que w_2 fos paraula de menor mida complint aquestes coses.

Exemple d'aplicació:



L_1 = llenguatge de les paraules acceptades per l'autòmat si començo l'execució des de l'estat 1.

L_2 = ... si comencem des de l'estat 2.

unió llenguatge que conté la paraula buida

$$L_1 = 0L_1 + 1L_2 + \Lambda$$

$$L_2 = 0L_1 + 1L_2 \rightarrow L_2 = \underset{\underset{A}{\shortparallel}}{1} L_2 + \underset{\underset{B}{\shortparallel}}{0 L_1} \rightarrow X = AX + B$$

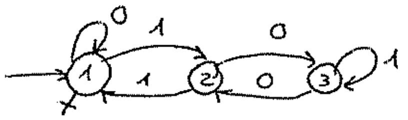
$$X = A^*B, \; L_2 = A^*B = 1^*0L_1$$

$$L_1 = 0L_1 + 11^*0L_1 + \Lambda \qquad CD \cup ED = (C \cup E)D$$

$$L_1 = \underbrace{(0+11^*0)}_{A} L_1 + \underset{B}{\Lambda}$$

$L_1 = A^*B = (0+11^*0)^*\Lambda$ Expressió regular que representa els múltiples de 2.

Trobar expressió regular pels múltiples de 3:



defineixo L_1, L_2 i L_3

Tots estats finales tenen $+\Lambda$

$$L_1 = 0L_1 + 1L_2 + \Lambda$$

$$L_2 = 1L_1 + L_2 + 0L_3 \longrightarrow L_2 = L_2 + 1L_1 + 0L_3$$

$$L_3 = 0L_2 + 1L_3 \longrightarrow L_3 = \underset{\underset{A}{\shortparallel}}{1} L_3 + \underset{\underset{B}{\shortparallel}}{0 L_2} \longrightarrow \boxed{X = AX + B}$$

$$X = A^*B, \; L_3 = A^*B = 1^*0L_2$$

$$\left\{ \begin{array}{l} L_1 = 0L_1 + 1L_2 + \Lambda \\ L_2 = 0L_1 + L_2 + 01^*0L_2 \\ L_3 = 1^*0L_2 \end{array} \right.$$

$$L_1 = 0L_1 + 1L_2 + \Lambda$$

Poden quedar expressions regulars diferents

$$L_2 = 0L_3 + 1L_1$$

$$L_3 = 0L_2 + 1L_3$$

$$L_3 = 0(0L_3 + 1L_1) + 1L_3 = 00L_3 + 01L_1 + 1L_3 = \underset{\underset{A}{\shortparallel}}{(00+1)} L_3 + \underset{\underset{B}{\shortparallel}}{01 L_1}$$

$$L_3 = A^*B = (00+1)^*01L_1$$

✱ sempre conté λ

$L_1 = 0L_1 + 1(0((00+1)^*01L_1) + 1L_1) + 1L_1) + \Lambda$

$L_1 = (0+10(00+1)^*01+11)L_1 + \Lambda$

$L_1 = (0+10(00+1)^*01+11)^*\Lambda$

També es pot obtenir

$L_1 = (\underset{\downarrow}{0} + \underbrace{1(01^*0)^*1}_{\downarrow})^*\Lambda$

queda a l'estat inicial

em porta a estat inicial

1 vaig a ②, $(01^*0)^*$ totes execucions possibles d'anar des de ② a ③ i tornar, després amb 1 torno a ①

CONCLUSIÓ: tot llenguatge regular és representable amb una expressió regular.

Com a cas particular del que hem estat veient, es pot justificar que la classe dels llenguatges regulars està inclosa en la classe dels incontextuals: cada autòmat té una gramàtica equivalent.

$L_1 \rightarrow 0L_1 \mid 1L_2 \mid \lambda$

$L_2 \rightarrow 0L_3 \mid 1L_1$

$L_3 \rightarrow 0L_2 \mid 1L_3$

formada sobre terminals

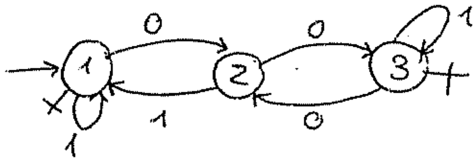
A partir d'autòmats sempre es generen gramàtiques lineals.

Una gramàtica és lineal dreta si totes les seves produccions són de la forma $X \rightarrow wY$ on w és terminal, i és lineal esquerra si totes les produccions són de la forma $X \rightarrow Yw$.

Per generar una gramàtica lineal esquerra, es procedeix de forma diferent però similar:

- redefinim L_1, L_2, L_3 així:

L_1 = paraules que em porten des de l'estat inicial fins l'estat 1.



L_2 = paraules que em porten fins l'estat 2.

L_3 = paraules que em porten fins l'estat 3.

← des de 1 fins 2 i després ve 1

$L_1 \longrightarrow L_2 1 \mid L_1 1$ → dés d'on ha vingut ??

$\quad\quad \mid \lambda$

$L_2 \longrightarrow L_1 0 \mid L_3 0$

$L_3 \longrightarrow L_2 0 \mid L_3 1$

inicial $S \longrightarrow L_1 \mid L_3$

Existeixen llenguatges incontextuals que no són regulars. Per exemple $\{a^n b^n\}$

No es pot reconèixer amb un autòmat

$S \rightarrow aSb \mid \lambda$ No és regular

20-5-05

Exemple de llenguatge <u>no regular</u>:

~~↳ no hi ha autòmat que ho reconegui~~

~~+ o s que el n. d'estats, al passar la 2a vegada han perdut el compte.~~

~~$\{a^n b^n\}$~~

~~$S \longrightarrow aSb \mid \lambda$~~

~~Suposem que $\{a^n b^n\}$ és regular. Llavors existeix un autòmat (DFA) A que reconeix aquest llenguatge. Sigui N el nombre d'estats d'A. L'execució d'A llegint a^N passa per N+1 estats, i com que només n'hi ha n, hi ha hagut una repetició d'estat. Per tant existeixen i, j tals que $q_0 a^i = q_0 a^i a^j$ on $j \geq 1$ (una a de més).~~

~~La paraula $a^i b^i$ és acceptada, i per tant $q_0 a^i b^i$ és estat acceptador. Aquest estat és també $(q_0 a^i) b^i = (q_0 a^i a^j) b^i$, així que la paraula $a^{i+j} b^i$ també s'accepta, que és una CONTRADICCIÓ amb la suposició que A reconeix $\{a^n b^n\}$.~~

20 Maig 05.

Exemple:

- Exemple de llenguatge no regular:

  $\{a^n b^n\}$

  sí és incontextual: $S \to aSb \mid \lambda$.

* Demostrem que el llenguatge no és regular per reducció al absurd:

Suposem que $\{a^n b^n\}$ és regular. Llavors existeix un autòmat (DFA) que reconeix $\{a^n b^n\}$.

Sigui N el nombre d'estats necessaris per reconèixer L en A. L'execució de l'autòmat A llegint a^N passa per N+1 estats, i com que només hi ha N, hi ha hagut una repetició d'estat. Per tant, existeixen i, j tals que $q_0 \cdot a^i = q_0 a^i a^j$ on $j \geq 1$.

La paraula $a^i b^i$ és acceptada, i per tant $q_0 a^i b^i$ és ESTAT ACCEPTADOR on aquest estat també és $(q_0 a^i) b^i = (q_0 a^i a^j) b^i$, així doncs la paraula $a^{i+j} b^i$ també s'accepta i aquesta paraula no pertany al llenguatge, doncs arribem a una contradicció amb la suposició que A reconeix $\{a^n b^n\}$.

D'aquí podem deduir que existeix un lema per demostrar la no-regularitat dels llenguatges.

= Lema de Bombament:

El següent és una condició suficient per justificar que un llenguatge L donat no és regular.

Cal comprovar que:

$\forall N \quad \exists w \in L$ tal que:

1. $|w| \geq N$
2. Per tota descomposició $w = xyz$ complint $|y| \geq 1$, $|xy| \leq N$ existeix i tal que $xy^i z \notin L$.

Aplicació en l'exemple anterior amb $L = \{a^n b^n\}$:

$\{a^n b^n\}$ és no-regular i ho demostrem usant el Lema de Bombament.

Demostració:

Fixem una N qualsevol escollim $w = a^N b^N$ que pertany a L i $|w| \geq N$. Sigui $w = xyz$ una descomposició tal que $|y| \geq 1$ i $|xy| \leq N$. Necessàriament, x, y, z són de la forma $x = a^j$, $y = a^k$, $z = a^{N-j-k} b^N$, on $k \geq 1$.

Tenim que $xy^2 z \in L$, doncs $xy^2 z = a^j a^{2k} a^{N-j-k} b^N = a^{N+k} b^N$

El llenguatge és NO-regular.

◦ Exemple:

El llenguatge $\{a^i b^j \mid i \neq j\}$ és no-regular.

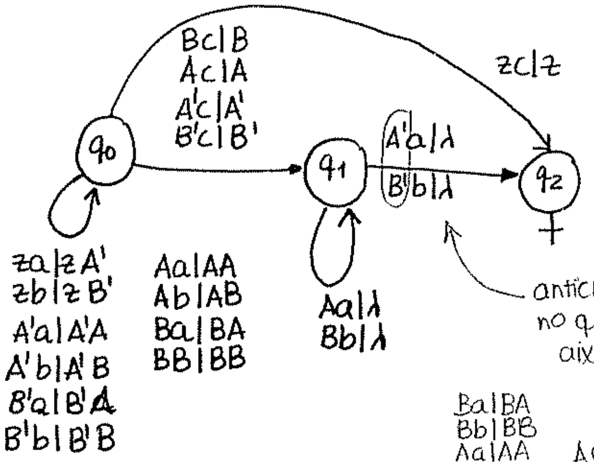
Sí ho demostrem usant el lema:

Fixat N, ens interessa agafar $a^N b^{N!}$

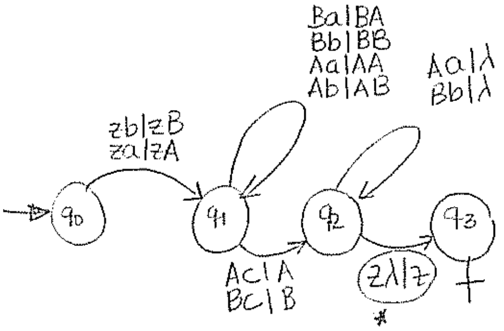
• PDA's

• $L = \{wcw^R \mid w \in (a+b)^*\}$

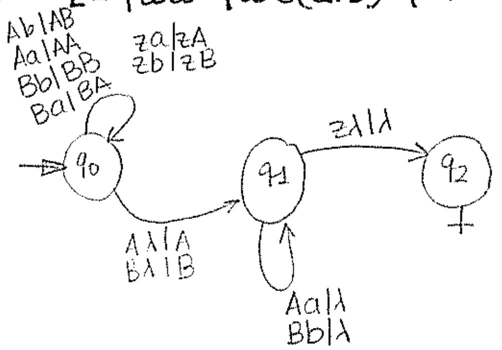
! Autòmats amb pila Indeterministes →
accepta un mot w quan existeix un camí acceptador, entre totes les tries possibles una seqüència que ens porta a la config. acceptadora.
Rebutja quan cap de les tries possibles porta a la config. acceptadora.



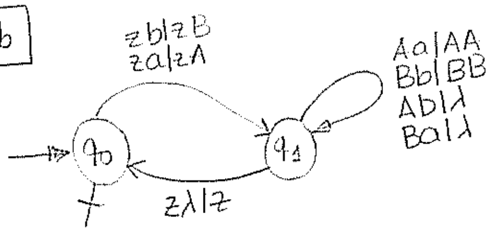
anticipem en un pas el coneixement que no queden més símbols útils a la pila, així evitem les λ-transicions *



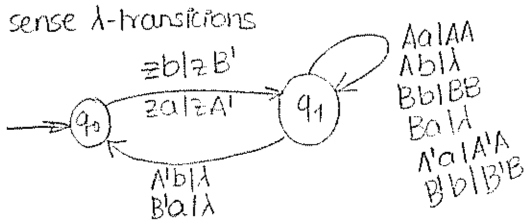
• $L = \{ww^R \mid w \in (a+b)^*\}$ ← palíndroms de long parella



• $|w|_a = |w|_b$



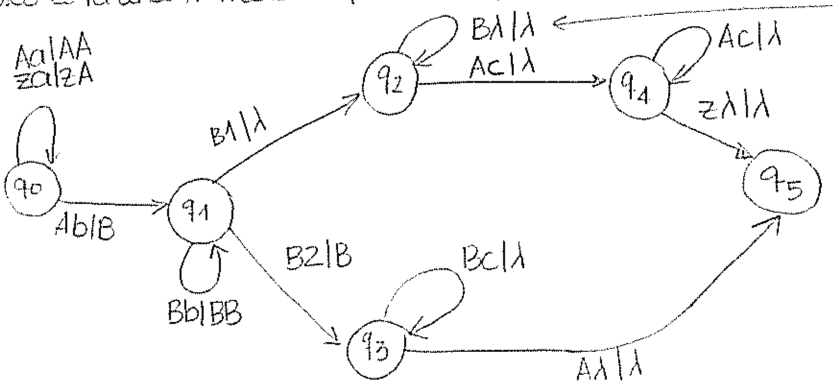
sense λ-transicions



• $L = \{a^nb^m1c^n \mid n,m \geq 1\} \cup \{a^nb^m2c^m \mid n,m \geq 1\}$

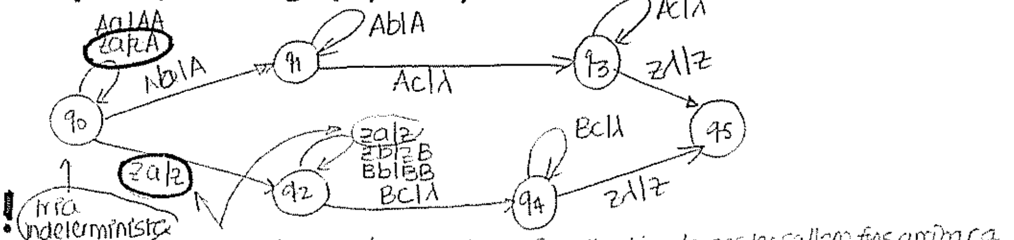
convertexen el llenguatge en determinista

En un primer moment apilem a pila totes les a's i b's per ordre, El símbol 1 o 2 que es llegeix força els 2 camins:
① Es desapilen amb λ-transicions les B's que hi han a la pila damunt les A's (com a mínim una → m≥1) i es realitza el comptatge.
② El comptatge es pot realitzar directament.
En els 2 casos es fa una λ-transició quan el comptatge acaba tenint èxit.



entra en bucle sense consumir símbols del mot d'entrada.

• $L = \{a^nb^mc^n \mid n\overset{m}{\geq}1\} \cup \{a^nb^mc^m \mid n,m \geq 1\}$



no apilem les a's q1 només ens fa falta b's c's ens les saltem fins arribar a la primera b → zb|zB

11.5. Format del fitxer d'informació sobre les dades.

El fitxer serà en format txt.
Contindrà la cruïlla d'inici i la destí, un volum de desplaçament per cada franja horària i dia, és a dir, un total de sis desplaçaments de ciutadans per la ciutat.

Parada_inici Parada_final D1 D2 D3 D4 D5 D6

D1 → Correspon al desplaçament de ciutadans que realitza algun moviment en un dia laborable entre les 21h – 7h
D2 → Correspon al desplaçament de ciutadans que realitza algun moviment en un dia laborable entre les 7h – 9h
D3 → Correspon al desplaçament de ciutadans que realitza algun moviment en un dia laborable entre les 9h – 18h
D4 → Correspon al desplaçament de ciutadans que realitza algun moviment en un dia laborable entre les 18h – 21h
D5 → Correspon al desplaçament de ciutadans que realitza algun moviment en dissabte o festiu entre les 21h – 13h
D6 → Correspon al desplaçament de ciutadans que realitza algun moviment en dissabte o festiu entre les 13h – 21h

23 Maig 05

* Resum conceptes bàsics

-Tenim una subrutina : interpreta(p)

$\underset{\substack{\uparrow \\ \text{sortida del} \\ \text{programa p}}}{s} = \text{interpreta}(p, \underset{\substack{\uparrow \\ \text{entrada.}}}{e})$

$\underset{\substack{\uparrow \\ \text{sortida} \\ \text{si para}}}{s} := \text{interpreta}(p, e, \#\text{passos})$

- A cada nombre nombre natural li podem fer correspondre un programa.
- programa : funció $f : \mathbb{N} \rightarrow \mathbb{N}$.
- Podem representar un programa amb TM

```
entrada x
    interpreta(M, x)
    donem sortida 1
```

màquina que calcula la suma dels resultats de M1 i M2

```
entrada x
    donem sortida $\underbrace{M_1(x)+M_2(x)}$
                  interpreta$(M_1,x)$
                  + intepreta$(M_2,x)$
```

== Suposem que M_1 i M_2 són màquines/programes que calculen predicats, és a dir, verifiquen una propietat sobre l'entrada i donen sortida 1, si aquesta es compleix i 0 si no es compleix
M_1 calcula la propietat P_1 és equivalent a dir $\forall x$:

- predicat: funció que va a parar a la parella $\{0,1\}$ $<\begin{matrix} 1 \text{ Compleix Propietat} \\ 0 \ \underline{\text{NO}} \end{matrix}$

M_1 dóna sortida 1 amb entrada x } si $\varphi_1(x) = 1$

M_1 dóna sortida 0 amb entrada x } si $\varphi_1(x) = 0$.

→ construïm programes que calculin $P_1 \wedge P_2$, $P_1 \vee P_2$, $\neg P_1$
Feu el mateix si ara parlem de què M_1 i M_2 calculen semi-predicats.
o bé donen sortida 1 o bé no s'aturen.

```
M1 :  Entrada x
      simular M amb entrada x
      Escriure 1.

M2 :  Entrada x
      simular M amb entrada x
      Escriure 1.
```

```
M1 ∧ M2 :
      Entrada x
      Simular M1 amb entrada x
      Simular M2 amb entrada x
      Escriure 1.
   ≡
      Entrada x
      dóna sortida
        interpretar(M1,x) ∧
        interpreta(M2,x)
```

```
M1 ∨ M2 :
   Entrada x
   dóna sortida interpreta(M1,x) ∨
                interpreta(M2,x)
≡
   Entrada x
   z = simular M1 amb entrada x
   si z és terminal → escriu 1
   sino
      simular M2 amb entrada x
      Escriu 1
```

↓ Error : si M_1 no para, llavors no podrem avaluar M_2 i podria ser que M_2 si parés.

```
¬P1 :  Entrada x
       Donar
       sortida ¬interpreta(M1,x)
```

⟹ Solució ⟹

```
Entrada x
atura$_1$:= fals
atura$_2$:= fals
npassos:= 1

mentre no atura$_1$ $\wedge$ no atura$_2$ fer

    <s$_1$, atura$_1$>:= interpreta (M$_1$, x, npassos)
    <s$_2$, atura$_2$>:= interpreta (M$_2$, x, npassos)
     npassos++;
fmentre
Escriure 1
```

FUNCIÓ CREIXENT:
una funció parcial $f: \mathbb{N} \to \mathbb{N}$ és creixent
si per tot parell $i<j$ la funció definida
és $f(i)<f(j)$

* Fer programes que :

③• A partir d'un programa d'entrada, doni sortida 1 si aquesta entrada no computa una funció creixent (estrictament creixent)

①• A partir d'un programa d'entrada, doni sortida 1 si el programa d'entrada s'atura amb alguna entrada.

②• A partir d'un programa d'entrada, doni sortida 1 si el d'entrada dóna sortida 5 amb alguna entrada.

①. $L = \{ x \mid \exists y \; M_x(y)\downarrow \}$

$L(M_2) = L$ llenguatge que reconeix M_2 és L

```
Entrada x
npassos := 1
atura:=fals
mentre no atura fer
    desde ω:=0 fins npassos i mentre no atura fer
        <i, atura> := interpreta (x, ω, npassos)
        ② atura = atura $\wedge$ (i = 5)
    fdesde
    npassos++;
fmentre
donar sortida 1
```

③.

```
Entrada x
npassos:=1
creixent:=cert
mentre creixent fer
    sant := -∞
    desde entr =0 fins npassos $\wedge$ mentre creixent fer
        < s, atura> := interpreta(x, entr, npassos)
        si atura llavors
            si s ≤ sant llavors creixent := fals
            sino
                sant:=s
            fsi
        fsi
    fdesde
    npassos++
fmentre
Escriure 1
```

④ Donats 2 programes d'entrada, donar sortida 1 si hi ha alguna entrada amb la que els 2 s'aturen i donen el mateix resultat.

REDUCCIONS :

Suposem que hi ha una subrutina que anomenem K que sempre s'atura i dóna cert si i només si el programa x que li donem com a entrada, s'atura amb entrada ell mateix.

$$K = \{x \mid H_x(x)\downarrow\}$$

* Fem un programa que ens digui si un programa d'entrada donat, s'atura amb entrada 5.

```
MX
Entrada x
    p:= [ entrada y
          interpreta(X,5)
    llavors si p↓ avaluem
    s:=K(p)
```

programa codificat amb la variable p

entrada qualsevol

$Hx(5)\Downarrow$

respón a la pregunta inicial!

Hem construit p de manera que: $\boxed{\varphi_p(p)\downarrow \Leftrightarrow \varphi_x(5)\downarrow}$

* Fer un programa que digui si el programa d'entrada és creixent

```
Entrada x
    p:=   Entrada y
```

Suposem que tenim una subrutina que decideix si un programa d'entrada s'atura amb entrada 5, i utilitzant-la resolem el problema K.

```
Entrada x
p:= [ Entrada y
      si y=5 → interpreta(x,x)

s:=subrutina(p)
Donar sortida s
```

Això resol K

p s'atura amb entrada 5

$\Updownarrow$

x s'atura amb entrada 5

26 Maig 05

E1. Veure que són indecidibles els següents problemes:

(a) Donat x, saber si φ_x és creixent.
Aquí creixent significa: f creixent si $\forall i<j$: $f(i), f(j)$ definides i $f(i)<f(j)$
(b) Donat x saber si φ_x és exhaustiva (Tothom té antiimatge, totes les sortides tenen entrada)
(c) Donat x saber si φ_x és total (definida x tota entrada)
(d) Donat x,y saber si $\varphi_x(y)=\varphi_y(x)$
(e) Donat x,y saber si $\varphi_x=\varphi_y$
(f) Donats x,y, saber si $\varphi_x>\varphi_y$ i φ_x, φ_y totals, és a dir, $\forall z: \varphi_x(z)\downarrow$ i $\varphi_y(z)\downarrow$ i $\varphi_x(z)>\varphi_y(z)$

E2. Definim Màquina de Turing acotada com una TM que només treballa amb tanta cinta com la pròpia entrada (no pot moure el capçal més enllà de l'entrada i sempre té marca d'inici i final d'entrada).

- Justificar que donada una TM acotada M i una entrada x, és decidible saber si M(x) s'atura.
- Justificar que és indecidible saber si donada una TM acotada M, existeix alguna entrada per la qual s'atura.

E3. Definim 2-DFA-3 com els autòmats finits bidireccionals amb 3 capçals (fem 3 lectures cada vegada, podem moure en conseqüència cada capçal a dreta o esquerra, i hi ha marca inicial i final d'entrada)
Veiem que:

- És decidible saber si, donat un 2-DFA-3 A i una entrada x, A accepta x.
- És indecidible saber si, donat un 2-DFA-3 A, existeix alguna entrada per la qual accepta (reduir des de PCP)

E1. a) Suposem que és decidible.
Llavors existeix M que decideix si una entrada x és funció creixent.
Utilitzant M construiré una màquina que decideix K.

```
Entrada x
    p: { Entrada y
         Mx(x)
         sortida y
sortida M(p)
```

Fer un programa que computi una funció creixent només si $\varphi_x(x)\downarrow$

- Total si només si $\varphi_x(x)\downarrow$, x és en realitat una constant del programa.

pot ser que no s'aturi/o sí

Hem trobat una màq que decideix K => contradicció

resol K

Total i creixent $\Rightarrow \varphi_x(x)\downarrow$
$M(p)\downarrow$

$M(p)\not\downarrow \Rightarrow$ és total i creixent

construir p no es penja, el que podria no aturar-se és la seva execució, però aquesta no es produeix mai, tant sols guardem la seva codif en p
Volem un programa que s'aturi sempre i retorni cert si la cod pertany o no. decideix x algun mètode si és creixent o no

$\Rightarrow$ P no et decideix K xq podria ser que no s'aturés $Mx(x)\uparrow$ ja que K representa un algorisme-conjunt tal que sempre $Mx(x)\downarrow$, llavors executem M amb la codif de p i aquesta és d'aturada segura (suposició) i sempre accepta K $\Rightarrow$ Aquesta màq decideix K
$\Rightarrow$ CONTRADICCIÓ $\Rightarrow$ INDECIDIBLE

27-Maig05

E4. Veieu que són indecidibles:

a.) Donat x, saber si $|\text{Dom}\,\varphi_x| = 3$.

b). Donat x, saber si φ_x és injectiva.

- f parcial és injectiva si $\forall x,y \in \text{Dom}\,f$ i $x \neq y$ passa que $f(x) \neq f(y)$

c). Donat x, saber si $\text{Im}\,\varphi_x \leq \text{Dom}\,\varphi_x$.

E4.a). És indecidible saber si $|\text{Dom}\,\varphi_x| = 3$.

Dom φ_x → Domini entrades màquina,
Tenim funció definida per 3 valors
d'entrada.
Suposem que és decidible i M és la màquina que ho fa.

$\varphi_x = x^2$ Domini (φ_x) = Els valors que pot prendre x.

decide φ_x
K

```
Entrada x
    p: Entrada y
        si y=0 v y=1 v y=2 → Mx(X) sortida 1
        sino ↑
    si M(p) té solució → Escriure(1)
    sino Escriure(0)
```

Versió prof: Mx(X)

```
si y=0 | y=1 | y=2 → donar sortida 1
sino aturem sense donar sortida   while(1){}
```

```
p: Entrada
    si y≥0 v y≤2
        sortida 1
    sino Mx(X)
        llavors sortida 1
si M(p)=1 llavors sortida 0
sino sortida 1
```

Si satura Dom és tot sinó 3

Reducció al complementari

E4.b) φ_x injectiva les imatges són diferents

Pensem en el negat ⇒ si $Mx(X)\downarrow$ ⇒ !injectiva.

```
Entrada x
    p: Entrada y
        Mx(X)
        sortida 0
    si M(p)=1 sortida 0     (dóna solució)
    sino sortida 1
```

Complementat

Alternativa:

```
Entrada x
    p: Entrada y
        Executar Mx(x) y passos
        si no para → sortida y
        sino → sortida 0
    si M(p)=1 sortida 0
    sino sortida 1
```

- saber si φ_x no és injectiu és E.R

```
Entrada x
    p:=0
    mentre cert fer
        des de y=0 fins p
            executar Mx(y) p passos
            si para →
                des de y'=y+1 fins p
                    executar Mx(y') p passos
                    si para i dóna la mateixa sortida
                    que Mx(y) → sortida 1.
                fdesde
        fsi
    fdesde
    p++; fmentre
```

26Mar605

E1.b) Suposem que és decidible.
Llavors existeix M que decideix si una entrada x és funció exhaustiva. (Tot element té antiimatge)
Utilitzant M construim una màquina que decideix K

p ha de ser una funció exhaustiva si $Mx(x)\Downarrow$

Alternativa

```
p: Entrada y
    si y>0
        sortida y
    sino
        Mx(x)
        sortida 0
    fsi
```

```
Entrada x
p: [ Entrada y
     Mx(x)
     sortida y
si M(p) dóna sortida → Escriure (1)
sino → Escriure(0)
```

✓ Decideix K: contradicció! $y \to f(y)$, $f^{-1}(y)$

(a) si $x \in K$ llavors $Mx(x)\Downarrow$ i per tant el p construït compleix $Mp(y)=y$ $\forall y$. Així que $M(p)$ és funció exhaustiva, de manera que $M(p)$ dóna sortida 1, i per tant el nostre programa dóna 1

(b) si $x \notin K$, $Mx(x)\uparrow$ i per tant el p construït compleix $\forall y$ $Mp(y)\uparrow$, així que M_p no és funció exhaustiva, de manera que $M(p)=0$ i per tant, el programa dóna 0

E1.c) Idem a,b

E1.d) Donat x,y saber si $\varphi_x(y)=\varphi_y(x)$

suposem que és decidible.
Llavors existeix M que decideix si amb entrada x,y $\varphi_x(y)=\varphi_y(x)$
Utilitzant M construim una màquina que defineix K.

$f_x: y \to \square$

$\quad\quad\quad\quad\quad\ \|$

$f_y: xx \to \square$

$$M(\langle x,y\rangle)=\begin{cases}1 & \text{si } \varphi_x(y)=\varphi_y(x)\\ 0 & \text{Altrament}\end{cases}$$

```
Entrada x
p1: [ Entrada y
      Mx(x)
      sortida 0

p2: [ Entrada y
      sortida 0
```

← fixem un, amb sortida sempre 0 ⇒ llavors l'altre programa només donarà la mateixa sortida quan $\varphi_x(x)\Downarrow$

```
si M(p1,p2) dóna solució → Escriu (1)
sino Escriu (0)
```

↑ tenim dues entrades x,y

E1.e) Donat x,y saber si $\varphi_x=\varphi_y$ → per qualsevol entrada $f_x: ? \to \square$, $f_y: ? \to \square$
Justificació idem (E1.d)!
En aquest cas la funció identitat [entrada y / Mx(x) / sortida y] no ens serveix!

E1.f) Assumim que això és decidible.
Llavors existeix màquina M que ho decideix
Intento solucionar K utilitzant M.

```
{ Entrada x
    p1: entrada y
        Mx(X)
        sortida 1
    p2: Entrada y
        Sortida 0

  si M(<p1,p2>)=1
  donem sortida 1
  sino 0.
```

Decideix K
⇓
contradicció