

## LINGUAGGIO S

### ● VARIABILI

► INPUT:  $x_1, x_2 \dots$

► OUTPUT:  $y$

► LOCALI:  $z_1, z_2 \dots$

### ● ETICHETTE: $A_1, \dots, B_1, \dots$

### ● ISTRUZIONI

► INCREMENTO:  $v \leftarrow v + 1$

► DECREMENTO:  $v \leftarrow v - 1$

► SALTO (CONDIZIONATO): IF  $v \neq 0$  GOTO L

Tutte le variabili partono da 0

E.

$$h(x_1, x_2) = \begin{cases} x_1 - x_2 & x_1 \geq x_2 \\ \uparrow & \text{altrimenti} \end{cases}$$

$y \leftarrow x_1$

$z \leftarrow x_2$

[C] IF  $z \neq 0$  GOTO A  
GOTO E

[A] IF  $y \neq 0$  GOTO B  
GOTO A

[B]  $y \leftarrow y - 1$   
 $z \leftarrow z - 1$   
GOTO C

## FUNZIONI PARZIALI

$$f: D \subseteq \mathbb{N}^R \rightarrow \mathbb{N}$$

Diremo che è **TOTALE** se  $D = \mathbb{N}^R$

## PREDICATO

Il **PREDICATO** è una proprietà verificabile (VERO, FALSO)  
definita su  $\mathbb{N}^k$

E.  
Predicato d'ugualanza:  $P(x_1, x_2) \Leftrightarrow x_1 = x_2$

I predici possono essere rappresentati attraverso funzioni **TOTALI**

## MACRO DI SALTO

IF  $Q(V_1, \dots, V_k)$  GOTO L =  $Z \leftarrow Q_r(V_1, \dots, V_k)$   
IF  $Z \neq 0$  GOTO L

## FUNZIONE PARZIALMENTE CALCOLABILI

Una  $f$  (parziale) e' PARZIALMENTE CALCOLABILE se un S-programma che la calcola, cioe' restituisce il valore  $f(x_1, \dots, x_k)$  per tutti gli input  $(x_1, \dots, x_k)$  su cui  $f$  e' definita e NON termina per gli altri.

$f$  si dira' CALCOLABILE se e' totale e parzialmente calcolabile

## S PROGRAMMA

Un S programma e' una lista ordinata di ISTRUZIONI

Il numero di istruzioni e' detto LUNGHEZZA del programma

## STATO

Lo STATO di un S programma e' una serie di uguaglianze della forma  $V = m$  tante quante sono le variabili che include tale uguaglianza per ogni  $V$  che compare nel programma

## INSTANTANEA

Un' **INSTANTANEA** di un programma  $P$  e' una coppia  $(i, \sigma)$  dove  $1 \leq i \leq m+1$  con  $m$  lunghezza del programma e  $\sigma$  e' uno stato di  $P$

L' instantanea ci dira' qual' e' l' istruzione successiva da eseguire ( $i$ ) e il valore di tutte le variabili ( $\sigma$ ) in ogni momento

L' instantanea si dice **TERMINALE** se  $i=m+1$ , se non lo e' la sua successiva sara' un' instantanea  $(j, \tau)$  dove :

- ①  $\tau = \sigma$  e  $j = i+1$  (istruzione pigra)
- ② Se l'  $i$ -esima istruzione e' d' incremento  $\tau$  si ottiene sostituendo in  $\sigma$  l' uguaglianza  $V=m$  con  $V=m+1$
- ③ Se l'  $i$ -esima istruzione e' di decremento ovvero  $\tau = \sigma$  se  $\sigma$  contiene  $V=0$  altrimenti sostituiremo  $V=m$  con  $V=m-1$  e  $j = i+1$
- ④ Se l'  $i$ -esima istruzione e' IF  $V \neq 0$  GOTO  $l$  ovvero  $\tau = \sigma$  e
  - $j = i+1$  se  $V \neq 0$  in  $\sigma$
  - se esiste un' etichetta  $[l]$ ,  $j$  sara' l' istruzione che identifica l' etichetta

• se non esiste una etichetta  $[l]$ ,  $j = n+1$  (uscita)

E.s.

[A]  $X \leftarrow X - 1$   
 $Y \leftarrow Y + 1$   
IF  $X \neq 0$  GOTO A

$$f(x) = \begin{cases} 1 & x=0 \\ x & \text{altrimenti} \end{cases}$$

$S_0 = \{x=2, y=3\}$  oppure  $\sigma = \{x=5, y=0, z_3=7\}$  (non importa che non esista  $z_3$ ) ma **NON**  $\{x=0, z_2=1\}$  perché  $Y$  non ha uguaglianza oppure  $\{x=5, x=3, y=1\}$  perché  $X$  ha due uguaglianze.

Quel è l'istante successiva di  $(1, \sigma)$ ?

$(2, \tau)$  con  $\tau = \{x=4, y=0, z_3=7\}$  (decremento di  $x$ )

la successiva di  $(2, \tau)$  è  $(3, \alpha)$  con  $\alpha = \{x=4, y=1, z_3=7\}$  (incremento di  $y$ ).

Poi avremo  $(1, \alpha)$  perché abbiamo un'istruzione di salto

### CALCOLO DI UN PROGRAMMA

Si dice **CALCOLO** per un programma P una successione di istanze  $s_1, \dots, s_k, \dots$  in cui  $\forall i \geq 1, s_{i+1}$  è successiva di  $s_i$  e:

- la successione è infinita (calcolo **NON TERMINALE**)

- la successione è finita e l'ultimo termine della successione è un'istantanea terminale (colloco **TERMINANTE**)

### STATO INIZIALE

Dato un programma  $P$  e una  $K$ -pla  $(z_1, \dots, z_k) \in \mathbb{N}^K$ , definiamo **STATO INIZIALE** l'insieme

$$\left\{ x_1=z_1, \dots, x_k=z_k, y=0 \right\} \cup \left\{ V=0 \mid V \text{ occorre in } P \text{ e NON è in } X_1, \dots, X_k \right\}$$

In pratica le  $K$  variabili iniziali hanno valori definiti, le successive saranno tutte 0 (sia di input, che locali che di output)

L'**ISTANTANEA INIZIALE** è  $(1, 0)$  con 0 stato iniziale.

E.

Nel caso precedente per  $K=1$  e  $z=6$ , lo stato iniziale corrispondente di  $P_1$  è  $\{x_1=6, y=0\}$

Per  $K=3$  e  $(2, 1, 3)$ , lo stato iniziale è  $\{x_1=2, x_2=1, x_3=3, y=0\}$

Per  $K=0$  lo stato iniziale è  $\{x_1=0, y=0\}$

## FUNZIONE DI UN PROGRAMMA

$\forall K \in V (x_1, \dots, x_K) \in \mathbb{N}^K$  definiamo:

$$\Psi_p^{(K)}(x_1, \dots, x_K) = \begin{cases} y & \text{se } P \text{ ha un calcolo terminante che inizia con l' istantanea iniziale corrispondente a } (x_1, \dots, x_K) \text{ e l' istantanea terminale contiene } Y=y \\ \uparrow & \text{altrimenti} \end{cases}$$

*(n° variabili)*

Se abbiamo un' istantanea iniziale corrispondente a  $(x_1, \dots, x_K)$  il valore della funzione sarà il valore che la variabile di output all' istante terminante , altrimenti non è definita

### OSS

$K$  può essere maggiore del numero effettivo di variabili del nostro programma, in tal caso quelle in eccedenza vengono ignorate.

Se invece dovesse essere minore, i "mancati" sono posti a 0.

## FUNZIONE PARZIALMENTE CALCOLABILE

Una funzione  $f$  di  $K$  variabili si dice **PARZIALMENTE CALCOLABILE** se esiste un programma  $P$  tale che

$$f(x_1, \dots, x_K) = \Psi_p^{(K)}(x_1, \dots, x_K) \quad \forall (x_1, \dots, x_K)$$

Si noti che  $f$  dev'essere definita sui valori della funzione

### HACRO DI SOSTITUZIONE

Se  $f$  è parzialmente calcolabile, la macro  $V' \leftarrow f(V_1, \dots, V_k)$  sostituisce nel programma:

$$Z_m \leftarrow 0$$

$$Z_{m+1} \leftarrow V_1$$

...

$$Z_{m+k} \leftarrow V_k$$

$$Z_{m+k+1} \leftarrow 0$$

...

$$Z_{m+h} \leftarrow 0$$

$P_m$

}  $m^{\circ}$  di variabili che il programma usa per calcolare  $f$

Programma che calcola  $f$

$$[E_m] V' \leftarrow Z_m$$

output   input   variabili d'appoggio   etichetta d'uscita   etichette

dove  $f$  calcolata da  $P = P(Y, X_1, \dots, X_k, Z_1, \dots, Z_h, E, A_1, \dots, A_e)$

e  $P_m = P(Z_m, Z_{m+1}, \dots, Z_{m+k}, Z_{m+k+1}, \dots, Z_{m+k+h}, E_m, A_{m+1}, \dots, A_{m+e})$

con  $m$  intero abbastanza grande affinché  $Z_m, Z_{m+1}, \dots, Z_{m+k},$

,  $Z_{m+k+1}, E_m, A_{m+1}, \dots, A_{m+e}$  non compaiano altrove nel

programma che richiama la macro (creando conflitto)



E. pg 39 n°

$y \leftarrow x_1$   
[A] IF  $x_2 = 0$  GOTO E  
 $y \leftarrow y + 1$   
 $y \leftarrow y + 1$   
 $x_2 \leftarrow x_2 - 1$   
GOTO A

$$\Psi_p^{(1)}(x) = x$$

$$\Psi_p^{(2)}(x_1, x_2) = x_1 + 2x_2$$

$$\Psi_p^{(3)}(x_1, x_2, x_3) = x_1 + 2x_2 \quad (x_3 \text{ man c'è})$$

### COMPOSIZIONE

Sono  $f, g_1, \dots, g_k$  funzioni  $n$ -arie e  $h$   $k$ -aria tali che

$$\forall x_1, \dots, x_n \quad f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

Allora  $f$  si ottiene per **COMPOSIZIONE** da  $h$  e le  $g_i$ .

Se  $g_i$  e  $h$  **man** sono totali,  $f$  sarà definita per tutti e solo i valori

$x_1, \dots, x_n$  tali che

$$g(x_1, \dots, x_n) \downarrow, \dots, g_k(x_1, \dots, x_n) \downarrow \text{ e} \quad (\text{sono definiti})$$
$$h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)) \downarrow$$

## TEOREMA

la composizione di funzioni parzialmente calcolabili è parzialmente calcolabile

## DIMOSTRAZIONE

Supponiamo che le  $g_i$  e  $h$  siano parzialmente calcolabili e  $f$  si ottenga da esse per composizione. Allora scriviamo il nostro programma  $P$  che calcola  $f$ :

$$z_1 \leftarrow g_1(z_1, \dots, z_m)$$

...

$$z_k \leftarrow g_k(z_1, \dots, z_m)$$

$$y \leftarrow h(z_1, \dots, z_k)$$

è tale che  $\Psi_p^{(m)}(x_1, \dots, x_m) = f(x_1, \dots, x_m)$  quindi parz. calc. per definizione

5.

$d(x) = 2x$  è calcolabile, dato che  $d(x) = m(2, x)$  con  $m(x, x_2) = x_1 x_2$

calcolabile e  $f_2(x) = 2$  e  $i(x) = x$  anch'esse calcolabili.

Analogamente  $g(x) = 4x^2$  è calcolabile data da  $g(x) = m(d(x), d(x))$ .

Inoltre  $h(x) = 4x^2 - 2x$  è parzialmente calcolabile dal teorema, ma dato

che  $4x^2 \geq 2x \forall x$  è anche totale e dunque calcolabile

## RICORSIVE PRIMITIVE

Siano  $f$  una funzione unaria,  $K \in \mathbb{N}$  e  $g$  binaria totale tali che:

$$\begin{cases} f(0) = K & (\text{con } K \geq 0) \\ f(t+1) = g(t, f(t)) & \forall t \geq 0 \end{cases}$$

$f$  si ottiene da  $K$  e  $g$  per **RICORSIONE PRIMITIVA**

## OSS

In generale se  $f$  è  $(m+1)$ -aria con  $h$ -unaria e  $g$   $(m+2)$ -aria

$$\begin{cases} f(x_1, \dots, x_m, 0) = h(x_1, \dots, x_m) \\ f(x_1, \dots, x_m, t+1) = g(t, f(x_1, \dots, x_m, t), x_1, \dots, x_m) & \forall t \geq 0 \end{cases}$$

Allora  $f$  si ottiene per ricorsione primitiva da  $h$  e  $g$

E.

Sia  $f(x_1, x_2) = x_1 + x_2$ , allora

$$\begin{cases} f(x_1, 0) = x_1 = i(x_1) \\ f(x_1, t+1) = f(x_1, t) + 1 = g(t, f(x_1, t), x_1) \end{cases}$$

dove  $g(x_1, x_2, x_3) = x_2 + 1$

## TEOREMA

Sia  $f$   $(n+1)$ -aria ottenuta per ricorsione primitiva da  $h$   $n$ -aria  
e  $g$   $(m+2)$ -aria calcolabili, allora  $f$  è calcolabile

## DIMOSTRAZIONE

Scriviamo un programma che calcola  $f$  come funzione  $(n+1)$ -aria

$$y \leftarrow h(x_1, \dots, x_m)$$

[A] IF  $x_{m+1} = 0$  GOTO E

$y \leftarrow g(z, y, x_1, \dots, x_m)$   $z$  è l'output e  $y$  è il valore di  $f$

$$z \leftarrow z + 1$$

$$x_{m+1} \leftarrow x_{m+1} - 1 \quad \text{quando esce } z \text{ vale } X_{m+1} - 1$$

GOTO A

## CLASSE PRC

Un insieme di funzioni totali  $C$  è una CLASSE PRC se

- Contiene le funzioni INIZIALI ovvero:

►  $m(x) = 0$  funzione nulla

►  $s(x) = x + 1$  funzione incremento

►  $U_i^{(n)} = (x_1, \dots, x_n) = x_i \quad \forall n \text{ e } 1 \leq i \leq n$  proiezione

• è chiuso rispetto alla composizione e alla ricorsione primitiva ovvero se:

►  $f(x_1, \dots, x_m) = h(g_1(x_1, \dots, x_m), \dots, g_k(x_1, \dots, x_m))$  per qualche  $h \in$

$g_1, \dots, g_k$  appartenenti a  $C$ , allora  $f \in C$

$$\int f(x_1, \dots, x_m, 0) = h(x_1, \dots, x_m)$$

$$\left\{ \begin{array}{l} f(x_1, \dots, x_m, t+1) = g(t, f(x_1, \dots, x_m, t), x_1, \dots, x_m) \end{array} \right.$$

e  $g, h$  appartengono a  $C$  allora  $f \in C$

Ad esempio le funzioni calcolabili sono classi PRC

## RICORSIVE PRIMITIVE

la più piccola classe PRC contiene esattamente le funzioni iniziali e tutte le funzioni che da essa si possono ricavare applicando composizione e ricorsione primitiva un numero finito di volte.

Tali funzioni sono dette **RICORSIVE PRIMITIVE**

Per definizione esse sono contenute in **TUTTE** le classi PRC e quindi sono calcolabili

E.

- $h(x, y) = x + y$  SOMMA

- $m(x, y) = xy$  PRODOTTO

## DI MOSTRAZIONE

Dimostriamo che  $m$  e  $h$  sono ricorse primitive:

- $h(x, 0) = x = \mu_1^{(1)}(x)$  caso base

$$h(x, t+1) = h(x, t) + 1 = s(h(x, t)) = g(t, h(x, t), x)$$

dove  $g(x, y, z) = s(\mu_2^{(3)}(x, y, z))$  incremento proiezione

Quindi  $h$  è ricorsiva primitiva

●  $m(x, 0) = 0 = m(x)$  caso base

$m(x, t+1) = x(t+1) = \underline{x} \underline{t} + \underline{x} = h(\underline{x} t, \underline{x})$  caso  $(t+1)$ -esimo

$$= g'(t, \underline{m(x,t)}, \underline{x})$$

dove  $g'(x, y, z) = h(u_2^{(3)}(x, y, z), u_3^{(3)}(x, y, z))$  sostituire il secondo + il Terzo

### DECREMENTO

$$p(x) = \begin{cases} 0 & x=0 \\ x-1 & \text{altrimenti} \end{cases}$$

$$p(0) = 0$$

$$p(t+1) = t = u_1^{(2)}(t, p(t))$$

### SOTTRAZIONE MODIFICATA (totale)

$$x-y = \begin{cases} x-y & x \geq y \\ 0 & \text{altrimenti} \end{cases}$$

$$\begin{aligned} x-0 &= x = u_1^{(1)}(x) \\ x-(t+1) &= p(x-t) = \dots \end{aligned}$$

(se  $x < t+1$  vole comunque perché  $p(0)=0$ )

### ALTRI ESEMPI

●  $|x-y| = \underline{(x-y)} + \underline{(y-x)}$

basta usare solo la composizione

• Predicato  $\delta(x) = (x=0) = \begin{cases} 1 & x=0 \\ 0 & \text{altrimenti} \end{cases}$

e' ricorsiva primitiva dato che  $\forall x \quad \delta(x) = 1 - x$

•  $d(x,y) = (x=y) = \begin{cases} 1 & x=y \\ 0 & \text{altrimenti} \end{cases}$

perche'  $d(x,y) = \delta(|x-y|)$  composizione di  $\delta$  e modulo

### TEOREMA

Siamo  $P, Q$  predicati ( $n$ -ari) appartenenti a una classe PRC  $C$ , allora  $\neg P, \neg Q, P \wedge Q, P \vee Q$  appartengono a  $C$

### DIMOSTRAZIONE

Basta osservare che:  $\neg P(x_1, \dots, x_n) = \delta(P(x_1, \dots, x_n))$  e

$(P \wedge Q)(x_1, \dots, x_n) = P(x_1, \dots, x_n)Q(x_1, \dots, x_n)$  e

$(P \vee Q) = \neg(\neg P \wedge \neg Q)$  (de Morgan)

Si possono ottenere per composizione di ricorsive primitive

E.s.

$$(x \leq y) \Leftrightarrow \neg(y \leq x)$$

## DEFINIZIONE PER CASI DI FUNZIONE

Supponiamo  $g, h$  funzioni  $n$ -arie in una classe PRC  $C$  e  $P$  predicato  $n$ -ario in  $C$ . Allora la funzione

$$f(x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n) & se P(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & altrimenti. \end{cases}$$

appartiene a  $C$

## DIMOSTRAZIONE

$$f(x_1, \dots, x_n) = g(x_1, \dots, x_n)P(x_1, \dots, x_n) + h(x_1, \dots, x_n)\bar{a}(P(x_1, \dots, x_n))$$

Se  $P$  e' verificata il secondo termine sara' proprio  $h$  e il primo  $g$

## COROLLARIO $m+1$ cose

Se

$$f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n) & P_1(x_1, \dots, x_n) \\ \vdots & \vdots \\ g_m(x_1, \dots, x_n) & P_m(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & altrimenti \end{cases}$$

e  $g_1, \dots, g_m, h, P_1, \dots, P_m \in C$ , allora  $f \in C$

## DIMOSTRAZIONE

Si dimostra per induzione:

- Definiamo  $h'(x_1, \dots, x_n) = \begin{cases} g_m(x_1, \dots, x_n) & P_m(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \text{altrimenti} \end{cases}$

- Allora  $h' \in C$ , e
- $$h(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n) & P_1(x_1, \dots, x_n) \\ \vdots & \vdots \\ g_{m-1}(x_1, \dots, x_n) & P_{m-1}(x_1, \dots, x_n) \\ h'(x_1, \dots, x_n) & \text{altrimenti} \end{cases}$$

Per induzione  $f \in C$

## PROPOSIZIONE

Sia  $f$  funzione  $(n+1)$ -aria appartenente a  $C$  classe PRC, allora

$$g(y, x_1, \dots, x_n) = \sum_{t=0}^y f(t, x_1, \dots, x_n)$$

$$h(y, x_1, \dots, x_n) = \prod_{t=0}^y f(t, x_1, \dots, x_n) \quad \text{PRODUTTORIA}$$

appartengono a  $C$

## DIMOSTRAZIONE

Sfruttiamo la composizione

$$g(0, x_1, \dots, x_n) = f(0, x_1, \dots, x_n)$$

caso base

$$g(y+1, x_1, \dots, x_n) = g(y, x_1, \dots, x_n) + f(y+1, x_1, \dots, x_n)$$

somma fino a y      valore di y+1

Analogamente vale per h

## QUANTIFICATORI

Sia P predicato  $(n-1)$ -ario in C classe PRC

Allora sono in C anche:

$$Q(y, x_1, \dots, x_n) = (\exists t \leq y) P(t, x_1, \dots, x_n)$$

QUANTIFICAZIONE ESISTENZIALE LIMITATA

$$R(y, x_1, \dots, x_n) = (\forall t \leq y) P(t, x_1, \dots, x_n)$$

QUANTIFICAZIONE UNIVERSALE LIMITATA

## DI MOSTRAZIONE

$$Q(y, x_1, \dots, x_n) \Leftrightarrow \sum_{t=0}^y P(t, x_1, \dots, x_n) \neq 0$$

(almeno una proposizione  
è verificata)

$$R(y, x_1, \dots, x_n) \Leftrightarrow \prod_{t=0}^y P(t, x_1, \dots, x_n) \neq 0$$

tutti sono verificati

## PROPOSIZIONE

$$\bullet (\exists t \leq y) P(t, x_1, \dots, x_m) \Leftrightarrow \sum_{t=0}^y P(t, x_1, \dots, x_m) \neq 0$$

$$\bullet (\forall t \leq y) P(t, x_1, \dots, x_m) \Leftrightarrow \prod_{t=0}^y P(t, x_1, \dots, x_m) \neq 0$$

E.

$$y|x \Leftrightarrow (\exists t \leq x) y \cdot t = x$$

$$\text{PRIHE}(x) \Leftrightarrow (\forall t \leq x) (t=1 \vee t=x \vee \neg(t|x)) \wedge x > 1$$

## MINIMALIZZAZIONE LIMITATA

Sia  $P$  predicato  $(n+1)$ -ario in  $C$  classe PRC e  $(\exists t \leq y) P(t, x_1, \dots, x_m)$

$$\text{Definiammo } g(y, x_1, \dots, x_m) = \sum_{u=0}^y \prod_{t=0}^u \mathbb{1}(P(t, x_1, \dots, x_m))$$

Allora  $g(y, x_1, \dots, x_m)$  e' il minimo  $t_0 \leq y$  tale che  $P(t_0, x_1, \dots, x_m)$  e' verificato

## DI MOSTRAZIONE

$$\prod_{t=0}^u \mathbb{1}(P(t, x_1, \dots, x_m)) = \begin{cases} 1 & u < t_0 \\ 0 & \text{altrimenti} \end{cases} \quad (t_0 \text{ e' il minimo che verifica } P)$$

$$\text{Quindi } g(y, x_1, \dots, x_m) = \sum_{u=0}^y \prod_{t=0}^u \mathbb{1}(P(t, x_1, \dots, x_m)) = t_0 \quad \text{sommerai tutti 1}$$

quanti sono i valori che non verificano  $P$  che sono proprio  $t_0$

(i numeri da 1 a  $t_0$  sono  $t_0$ ) quindi è verificata l'espressione

Se  $P$  non è mai verificata g varrebbe y, ciò non ci interessa per la definizione per casi che diamo ora.

### MINIMIZZAZIONE LIMITATA DEFINITA PER CASI

Definiamo allora la funzione di MINIMIZZAZIONE LIMITATA

$$\min_{t \leq y} P(t, x_1, \dots, x_m) = \begin{cases} g(y, x_1, \dots, x_m) & (\exists t \leq y) P(t, x_1, \dots, x_m) \\ 0 & \text{altrimenti} \end{cases}$$

tal funzione è ancora in  $C$  poiché è definita per caso e la distinzione tra i casi è data da un predicato in  $C$  (quantificazione esistenziale limitata su  $P$ ) e la funzione  $g \in C$  poiché ottenuta da  $P$  applicando  $\lambda, \Pi, \Sigma$ .

Esempio:

$$\bullet \lfloor x/y \rfloor = \min_{t \leq x} ((t+1)y > x) \quad \text{divisione intera}$$

$$\bullet x \bmod y = x - (y \cdot \lfloor x/y \rfloor) \quad \text{resto}$$

## PROPOSIZIONE

Definiamo  $p_0 = 0$  e per  $m \geq 1$  sia  $p_m$  l'  $m$ -esimo numero primo.

Allora  $p_m$  è funzione ricorsiva primitiva di  $m$ .

Osserviamo che  $p_{m+1} = \min_{t \leq p_m! + 1} (\text{Prime}(t) \wedge (t > p_m))$

## DMOSTRAZIONE

Tra  $p_m$  e  $p_m! + 1$  c'è almeno un primo perché  $p_m! + 1$  da sempre resto 1 se diviso per  $p_{i+1}, \dots, p_m$  quindi non è multiplo di nessuno di essi.

Dunque è primo o è multiplo di un primo  $\geq p_m$ .

Per verificare che  $p_m$  sia ricorsiva primitiva definiscono le funzioni:

$$\bullet h(y, z) = \min_{t \leq z} (\text{Prime}(t) \wedge t \geq y)$$

che da il minimo primo compreso tra  $y+1$  e  $z$  se esiste, 0 altrimenti ed è ricorsiva primitiva poiché ottenuta tramite minimizzazione limitata di predicato ricorsivo primario.

$$\bullet \text{Sia } k(x) = h(x, x! + 1) \text{ anch' essa ricorsiva primitiva}$$

Allora  $p_0 = 0$  e  $p_{m+1} = k(p_m)$  il che dimostra per ricorsione primitiva che  $p_m$  è ricorsiva primitiva.

## MINIMIZZAZIONE NON LIMITATA

Dato un predicato  $(n+1)$ -ario  $P(y, x_1, \dots, x_n)$  definiamo  $\min_y P(y, x_1, \dots, x_n)$  come il minimo valore di  $y$  per cui  $P(y, x_1, \dots, x_n)$  se esiste, non definita altrimenti.

$$\min_y P(y, x_1, \dots, x_n) = \begin{cases} \min \{y \in \mathbb{N} \mid P(y, x_1, \dots, x_n)\} & \exists y \mid P(y, x_1, \dots, x_n) \\ \uparrow & \text{altrimenti} \end{cases}$$

In generale **NON** è ricorsiva primitiva anche se  $P$  lo è e anche quando la funzione risultante è totale

## PROPOSIZIONE

Se  $P$  è calcolabile, allora  $\min_y P(y, x_1, \dots, x_n)$  è parzialmente calcolabile

## DIMOSTRAZIONE

Per ripetere dimostriamo che  $\min_y P(x_1, \dots, x_n, y)$  è parzialmente calcolabile;  
vole lo stesso per la minimizzazione su altre variabili.

Scriuiamo il programma che calcola la funzione:

[A] IF  $P(x_1, \dots, x_n, y)$  GOTO E

$y \leftarrow y - 1$

GOTO A

## FUNZIONE DI PAIRING

"ANGOLETTO"

$$\langle x, y \rangle = 2^x (2y+1) - 1$$

E' una funzione BIETTIVA ricorsiva primitiva:

$$\forall z \in \mathbb{N} \exists! (x, y) \in \mathbb{N} \times \mathbb{N} \mid z = \langle x, y \rangle$$

## DI MOSTRAZIONE

$$z = \langle x, y \rangle \iff z + 1 = 2^x (2y + 1)$$

Ogni intero positivo  $z+1$  si scrive in modo univoco come prodotto di una di una di 2 e un numero dispari (bionale) quindi  $x$  e  $y$  sono definite univocamente

## OSS

- Le "inverse"  $l(z)$  e  $r(z)$  della funzione saranno:

$$l(z) = \min_{x \leq z} (\exists y \leq z) \langle x, y \rangle = z$$

che restituisce il più piccolo (abbiamo appena dimostrato che è unico)  $x$  tale che esiste un qualsiasi  $y$  che verifica la proprietà  $\langle x, y \rangle = z$

(analogamente vale per  $r$ )

Riassumendo  $\leftrightarrow$  c'è:

- Biiettiva di  $\mathbb{N} \times \mathbb{N}$  in  $\mathbb{N}$
- Ricorsiva primitiva
- Con inversa ricorsiva primitiva

E.s.

$$\bullet \langle 1, 1 \rangle = 2^1(2 \cdot 1 + 1) - 1 = 5$$

$$\bullet 23 = \langle x, y \rangle \rightarrow 23 + 1 = 2^x(2 \cdot y + 1) \Leftrightarrow 24 = 2^3(2 \cdot 1 + 1)$$

$$23 = \langle 3, 1 \rangle$$

### NUMERO DI GÖDEL di n-plo

$[a_1, \dots, a_n] = \prod_{i=1}^n p_i^{a_i}$  è detto numero di Gödel

Per il TFA ogni intero positivo è primo o prodotto di primi

Quindi se  $z > 0$  siamo  $\exists m \geq 1$  e una n-plo  $(a_1, \dots, a_n)$

tale che  $z = [a_1, \dots, a_n]$  (scadenza in fattori primi)

Per ogni  $n$  fissata il numero di Gödel definisce una funzione ricorsiva primitiva.

Se  $[a_1, \dots, a_n] = [b_1, \dots, b_m]$  con  $m \leq n$  allora:

$\forall i \leq n, a_i = b_i$  e  $b_i = 0$  per  $i > m$

## OSS

- La "funzione" numero di Gödel è biettiva
- 1 è il numero di Gödel della sequenza vuota

## INVERSA DEL NUMERO DI GOEDEL

Definiamo  $v_i(x) = \min_{t \leq x} (\gamma(p_i^{t+1} | x))$

Ristituisce l'esponente dell'  $i$ -esimo numero primo che divide  $x$

E.

$$(24)_1 = 3 \quad \text{perché } 2^3 | 24 \text{ però } 2^4 \nmid 24$$

$$(1)_i = 0 \quad \text{qualsiasi numero primo che divide 1 ha 0 come esponente}$$

Per definizione di minimizzazione limitata  $(0)_i = 0 \forall i$

Quindi la funzione è totale (definita anche su 0)

E. pg 54 m<sup>o</sup> 3

- Sia  $\Pi$  la funzione che conta i primi  $\leq x$ , il predicato  $\text{Primo}(x)$  è sicuro.

I predici possono essere visti come programmi che valgono 1 se il predicato è verificato, 0 altrimenti. Se contiamo i valori di  $\text{Primo}(t)$  avremo proprio il valore di  $\Pi(x)$ .

$$P(x) = \sum_{t=0}^x P_{\text{mine}}(t) \quad \checkmark$$

- pg 58 nº 1,5

④  $h(x) = \lfloor \sqrt{2}x \rfloor$  **NON** è una funzione composta perché non in  $\mathbb{N}$

Possiamo usare un operatore di minimizzazione limitata

Il problema ci dice che la funzione ci da  $m \leq \sqrt{2}x$  (avendo  $\lfloor \sqrt{2}x \rfloor$ )

$m \leq \sqrt{2}x \rightarrow m^2 \leq 2x^2$  questo ci semplifica molto le cose, equivale a

$$\sqrt{2} \times <(m+1) \rightarrow 2x^2 < (m+1)^2 \quad \text{essendo strettamente minore sono il minimo}$$

Quindi  $h(x) = \min_{m \leq 2x} (2x^2 < (m+1)^2)$  (operatore di minimizzazione limitata)  
 $\leftarrow$  limite superiore

- 5 Sia  $R(x, t)$  predicato ric. prim. dimostrare che  $g(x, y) = \max_{t \leq y} R(x, t)$  che vale Q se  $\neg (\exists t \leq y) R(x, t)$ , e' ric. prim.

l'operatore di massimizzazione limitata puo' essere espresso con la minimizzazione

Basta osservare che se  $t_0$  è il massimo valore  $\leq x$  tale che  $R(x, t_0)$ , allora  $y - t_0 = s$  è il minimo valore  $\leq y$  tale che  $R(x, y - s)$ .

Quindi se  $(\exists t \leq y) R(x, t)$  allora  $s = \min_{s \leq y} R(x, y - s)$  e quindi per definizione

$$t_0 = g(x, y) = y - \min_{s \leq y} R(x, y - s)$$

In generale,  $g(x, y) = \begin{cases} y - \min_{s \leq y} R(x, y - s) & \text{se } (\exists t \leq y) R(x, t) \\ 0 & \text{altrimenti} \end{cases}$

### • pg 62 n° 3

Dimostrare che l'algoritmo di calcolo dell'  $n$ -esimo numero di Fibonacci è ric. prim.

L'algoritmo prevede l'utilizzo di due valori precedenti, questo complica le cose. Ci aiuta tuttavia la funzione angioletto (ric. prim.)

$$F(0) = 0, \quad F(1) = 1, \quad F(m+1) = F(m) + F(m-1) \quad \text{per } m \geq 1$$

Definiamo  $F'(m) = \langle F(m), F(m+1) \rangle$  e dimostriamo che è ric. prim.:

$$F'(0) = \langle 0, 1 \rangle = 2^0 (2 \cdot 1 + 1) - 1 = 2 \quad (\text{l'importante è che ricavo un numero da una coppia})$$

$$\begin{aligned} F'(m+1) &= \langle F(m+1), F(m+2) \rangle = \langle F(m+1), \underline{F(m)} + \underline{F(m+1)} \rangle \\ &= \langle \underline{r(F'(m))}, \underline{l(F'(m))} + \underline{r(F'(m))} \rangle \quad r \text{ e } l \text{ sono ric. prim} \end{aligned}$$

Dunque  $F'$  è ric. prim e poiché  $\forall m \quad F(m) = l(F'(m))$  anche  $F$  lo è.

## TEOREMA

Una funzione è parz. calc. se e solo se è possibile ottenerla dalle funzioni iniziali attraverso un numero finito di applicazioni degli operatori di composizione, ricorsione primitiva e minimizzazione non limitata.

## NUMERO DI GÖDEL

Il numero di Gödel data una successione  $x = [x_1, \dots, x_n] = \prod_{i=1}^n p_i^{x_i}$  (e' ric. prim.)

le cui inverse parziali sono

$$(x)_i = \min_{t \leq x} (\gamma(p_i^{t+1} | x)) \quad \text{massimo esponente di } i \text{ che divide } x$$

$$lt(x) = \min_{i \leq x} ((x)_i \neq 0 \wedge (\forall j \leq i) (j \leq i \vee (x)_j = 0)) \quad \text{LUNGHEZZA}$$

(quanti numeri primi dividono  $x$ )

Es.

$$40 = 2^3 \cdot 3^0 \cdot 5^1 = [3, 0, 1]$$

$(40)_3 =$  la massima potenza del  $3^{\circ}$  numero primo che divide  $40 = 5^1 = 1$

$lt(40) =$  minimo  $n$  per cui esiste una m-pola di cui  $40$  è numero di Gödel = 3

## OSS

$$\bullet \forall a_1, \dots, a_m \quad ([a_1, \dots, a_m])_i = \begin{cases} a_i & i \leq m \\ 0 & \text{altri casi} \end{cases}$$

- $n \geq \text{Lt}(x) \Rightarrow [(x)_1, \dots, (x)_n] = x$

## CODIFICA DI ISTRUZIONI E PROGRAMMI

① Assegnano un valore a ogni variabile che compare:

$y, x_1, z_1, x_2, z_2, \dots, x_n, z_n$

0, 1, 2, 3, 4, --

② Analogamente con le etichette (partendo da 1)

$A_1, B_1, C_1, A_2, \dots$

1 2 3 4

③ Se  $I$  è un'istruzione, definiamo  $\#I = \langle a, \langle b, c \rangle \rangle$  (ci sono 3 numeri per indicare 1, la funzione angoletto a sinistra) tale che:

- $a$  è il numero dell'etichetta dell'istruzione (0 se non c'è un'etichetta)

- $c$  è il numero della variabile che compare in  $I$

- $b$  sarà

- ▶ 0 se  $I$  è pigra ( $v \leftarrow v$ )

- ▶ 1 se è di incremento ( $v \leftarrow v+1$ )

- ▶ 2 se è di decremento ( $v \leftarrow v-1$ )

- ▶  $m+2$  se è soltanto all'ultima etichetta  $L$  (IF  $V \neq 0$  GOTO  $L$ )

t.

- Sia  $I = (x \leftarrow x-1)$

Allora  $\#I = \langle 0, \langle 2, 1 \rangle \rangle = \langle 0, 2^2(2 \cdot 1 + 1) - 1 \rangle = \langle 0, 11 \rangle = 2^0(1 \cdot 11 + 1) - 1 = 22$

- $\#J = 37$ , definiamo  $J$

$$37 = 2^a(2 \langle b, c \rangle + 1) - 1 \rightarrow 38 = 2^a(2 \langle b, c \rangle + 1)$$

La massima potenza gli 2 che divide 38 è  $2^1$  quindi  $a=1$

$$a=1, \langle b, c \rangle = \frac{19-1}{2} = 9 \quad \text{riteniamo per } g$$

$$g = 2^b(2c+1) - 1 \rightarrow 10 = 2^b(2c+1) \Rightarrow b=1, c=2$$

Quindi  $J$  e': [A]  $z \leftarrow z+1$

### CODIFICA DEI PROGRAMMI

Se  $P$  è il programma dato da  $(I_1, \dots, I_m)$  definiamo  $\#P = [I_1, \dots, I_m] - 1$

Sottraiamo 1 perché i numeri di Gödel partono da 1 ma non vogliamo anche la codifica di 0 per garantire la unicità.

Per rimediare alla non unicità dei numeri di Gödel (perché alla successione considerata possiamo sempre aggiungere 0 e il suo valore non cambierà) stabiliscono che un S-programma non può finire

con  $\#I=0$  cioè (colcolondola)  $y \leftarrow y$

E.

$\#P=11$  sarà

$12 = [2,1]$  troviamo  $\#I=2$  e  $\#I=1$

$2 = \langle a \langle b, c \rangle \rangle \Rightarrow a=0, \langle b, c \rangle = 1 \Rightarrow b=1, c=0 \rightarrow y \leftarrow y+1$

$1 = \langle a' \langle b', c' \rangle \rangle \Rightarrow a'=1, b'=0, c'=0 \rightarrow [A] y \leftarrow y$

$P = \begin{array}{l} y \leftarrow y+1 \\ [A] y \leftarrow y \end{array}$  (valida perché etichettata)

E.

Sia  $P$  il programma

[A]  $x \leftarrow x - 1$

$y \leftarrow y + 1$

IF  $x \neq 0$  GOTO A

$$\#P = [\#I_1, \#I_2, \#I_3] - 1$$

$$\#I_1 = \langle a_1, \langle b_1, c_1 \rangle \rangle$$

$$\begin{aligned} a_1 &= 1 \quad (\text{perche' [A]}) \\ b_1 &= 2 \quad (\text{perche' decrescente}) \\ c_1 &= 1 \quad (\text{perche' } x_1) \end{aligned} \quad \left. \right\} \langle 1 \langle 2, 1 \rangle \rangle = \langle 1, 2^1 (2 \cdot 1 + 1) - 1 \rangle = 45$$

$$\#I_2 = \langle a_2, \langle b_2, c_2 \rangle \rangle \quad \text{con } a_2 = 0, b_2 = 1, c_2 = 0$$
$$= \langle 0, \langle 1, 0 \rangle \rangle = \langle 0, 1 \rangle = 2$$

$$\#I_3 = \langle 0, \langle 3, 1 \rangle \rangle = 46$$

$$\text{Quindi: } \#P = [45, 2, 46] - 1 = 2^{45} \cdot 3 \cdot 5^{46} - 1$$

## TEOREMA

Esistono funzioni **NON** parzialmente calcolabili.

## DIMOSTRAZIONE

Ordiniamo tutte le funzioni parz. calcolabili in una successione, in corrispondenza del numero dei programmi che le calcolano:

Sei  $P(m)$  il programma tale che  $\#(P(m)) = m$  per ogni  $m$

Allora la successione è  $\Psi_{P(0)}^{(1)}, \dots, \Psi_{P(m)}^{(1)}$

Costruiamo una tabella infinita guardando i valori delle funzioni della successione:

la prima riga sarà

$\Psi_{P(0)}^{(1)}(0) \quad \Psi_{P(0)}^{(1)}(1) \dots$

} valori calcolati dalla funzione numero 0

l' $m+1$ -esima riga sarà  $\Psi_{P(m)}^{(1)}(0) \quad \Psi_{P(m)}^{(1)}(1) \dots$

Dove se  $\Psi_{P(m)}^{(1)}(\infty)$  non è definita poniamo -1.

Chiamiamo  $T_{m,n}$  il contenuto della tabella in posizione  $(m,n)$

cioè  $T_{m,n} = \Psi_{P(m)}^{(1)}(n)$  se  $\Psi_{P(m)}^{(1)}(n) \downarrow$ , -1 altrimenti

e definiscono  $f(m) = T_{m,m} + 1$  (solo oggetti della diagonale)

Allora  $f$  è totale ma sicuramente non era presente in Tabella perché rispetto a qualunque riga della tabella ha almeno un valore distinto.

$$\text{Infatti } \forall m, f(m) = T_{m,m+1} = \begin{cases} \psi_{p(m)}(m) + 1 & \text{se } \psi_{p(m)}(m) + 1 \\ 0 & \text{altrimenti} \end{cases}$$

Quindi  $f$  **NON** è calcolabile (altrimenti starebbe in tabella)

### TESI DI CHURCH-TURING

Ogni procedura algoritmica può essere emulata da un S-programma  
(o macchina di Turing o equivalenti) cioè:

ogni funzione i cui valori possono essere calcolati algoritmicamente  
è (parzialmente) calcolabile.

In base a questa tesi il teorema precedente dice che  
esistono funzioni per le quali nessun algoritmo generale potrà mai  
fornirci il valore su ogni generico input.

### PROBLEMA DELLA FERMATA

$\text{HALT}(x,y) \Leftrightarrow$  il programma numero  $y$  termina su input  $x$

$$\Leftrightarrow \psi_{p(y)}^{(x)}(x) \downarrow$$

$\text{HALT}$  non è calcolabile quindi non è possibile in generale scrivere  
una procedura che dica se un dato programma termina su un dato input

## TEOREMA

Il predicato HALT non è calcolabile

## DIMOSTRAZIONE

Ragioniamo per assurdo, quindi sia HALT calcolabile, possiamo usarlo come macro. Sarebbe quindi valido:

[A] IF HALT(x,x) GOTO A

E' possibile individuare un numero per questo programma.

Sia  $\#(P) = y_0$  avremo.

$$\Psi_p^{(1)}(x) = \begin{cases} 0 & \neg \text{HALT}(x,x) \\ \uparrow & (\text{x non viene toccata}) \text{ e si ferma} \\ & \text{altrimenti} \\ & \text{va in loop} \end{cases}$$

e quindi  $\forall x, \text{HALT}(x,y_0) \Leftrightarrow \neg \text{HALT}(x,x)$

Ma scegliendo  $x=y_0$  avremo  $\text{HALT}(x,y_0) \Leftrightarrow \neg \text{HALT}(x,y_0)$  ASSURDO!

## CONGETURA DI GOLDBACH

Ogni numero pari  $> 2$  è somma di 2 primi

Ad oggi non c'è nota la veridicità di quest' enunciato,

pertanto non sappiamo se esiste mai termine un programma che calcoli

$$\min_n (\neg ((\exists x \leq 2n+4) (\exists y \leq 2n+4) (\text{Prime}(x) \wedge \text{Prime}(y) \wedge (2n+4 = x+y))))$$

## FUNZIONE UNIVERSALE

Definiamo  $\Phi^{(n)}(x_1, \dots, x_n, y) = \Psi_{P(y)}^{(n)}(x_1, \dots, x_n)$ ,

intendendo in particolare che  $\Phi^{(n)}(x_1, \dots, x_n, y) \downarrow = \Psi_{P(y)}^{(n)}(x_1, \dots, x_n) \downarrow$

(l'ultima variabile di  $\Phi$  indica quale programma stiamo calcolando)

La funzione universale calcola tutti i programmi p.z. calc.

## TEOREMA DI UNIVERSALITÀ

$\forall n$  la funzione  $\Phi^{(n)}$  è parzialmente calcolabile

## CODIFICA DEGLI STATI DI UN PROGRAMMA

Ricordiamo che per STATO di un programma  $P$  intendiamo una lista di assegnazioni del tipo  $V = n$ , che contiene esattamente un'assegnazione per ogni variabile  $V$  che compare in  $P$ .

Rappresenteremo gli stati come segue:

Sia  $V_m$  la variabile numero  $m$  ( $y = V_0, x_1 = V_1, z_1 = V_2, \dots$ )

Allo stato  $\{V_i = m_i \mid i \in I\}$  corrisponderà il numero

$[m_0, m_1, \dots]$  dove si intende  $m_j = 0$  se  $j \notin I$

Esempio

Lo stato  $\{y = 0, x_1 = 2, x_2 = 3\}$  avrà il numero  $[0, 2, 0, 3, \dots] = 2^0 \cdot 3^2 \cdot 5^0 \cdot 7^3 = 3087$

2, non compare  
↓  
variables d'input

## PROGRAMMA UNIVERSALE

Il **PROGRAMMA UNIVERSALE**  $\mu^{(n)}$  funziona come una sorta di traduttore estraeendo l'ultima variabile d'input per definire quale programma eseguire; a partire da questo sare' in grado di determinare il successore di qualunque sua istantanea non terminale.

Per leggibilita' useremo variabili locali e etichette usualmente non ammesse dal linguaggio S

$Z \leftarrow X_{mn} + 1$  in Z salviamo il valore dell'ultima variabile

l'input e sommiamo 1 perché Z e' un numero di Gödel

$K \leftarrow 1$  in K salviamo il numero della prossima istruzione

$S \leftarrow \prod_{i=1}^m (P_{2i})^{x_i}$

S rappresenta lo stato (numero di Gödel) corrente  
(l'espressione moltiplica le variabili pari per avere le  
variabili locali) e quindi anche la variabile d'output

$[C] \text{ IF } (k = Lt(z) + 1 \vee k = 0) \text{ GOTO F}$

abbiamo raggiunto l'istantanea terminale

$U \leftarrow \underline{\tau}(t_k)$

prendiamo la massima potenza  
del k-esimo primo che divide z  
prendendo proprio il numero della  
k-esima istruzione

(cioè  $c_{b,c}$  della k-esima istruzione)

IF  $l(U)=0$  GOTO N

(istruzione pigia non fare nulla)

$P \leftarrow P_{l(U)+1}$

$l(U) = c$  (+1 perché le variabili puntano da 0)

IF  $l(U)=1$  GOTO A

istruzione di incremento

IF  $\neg (P \neq S)$  GOTO N

il valore attuale di  $c=0$

IF  $l(U)=2$  GOTO M

istruzione di decremento

$K \leftarrow \min_{i \leq l(t)} (l(U) = l((t)_i) + 2)$

istruzione di salto

(numero dell'istruzione = 2 + etichetta)

GOTO C

Ripeti il ciclo

[A]  $S \leftarrow S \cdot P$

S è dato da un prodotto di primi,

GOTO N

se lo moltiplichiamo per P ci sarà

sicuramente un aumento del suo esponente

e di conseguenza di una variabile.

[H]  $S \leftarrow \lfloor S/P \rfloor$

Analogo di prima

[N]  $K \leftarrow K+1$

GOTO C

[F]  $S \leftarrow (S),$

l'output è salvato nella prima

parte di S visto come numero di Goedel

## NOTAZIONI PER $\Phi$

$$\overline{\Phi}^{(m)}(x_1, \dots, x_m, y) = \Psi_{\rho(y)}^{(m)}(x_1, \dots, x_m)$$

Negli esercizi una notazione equivalente è data da  $\overline{\Phi}_y^{(m)}(x_1, \dots, x_m)$

Anche  $\overline{\Phi}^{(1)}(x, y) = I(x, y)$

## PREDICATO STEP COUNTER

Definiamo il predicato

$$STP^{(m)}(x_1, \dots, x_m, y, t) \Leftrightarrow \rho(y) \text{ termina dopo al più } t \text{ passi con l'input } x_1, \dots, x_m$$

Questo predicato è ricorsivo primitivo  $\forall m \geq 1$

## FUNZIONE DI ACKERMAN-PÉTER

La funzione di Ackermann-Péter è calcolabile ma **NON** ricorsiva primitiva

$$A(x, y) = \begin{cases} y+1 & x=0 \\ A(x-1, 1) & y=0 \\ A(x-1, A(x, y-1)) & \text{altri casi} \end{cases}$$

E.s.

$$\begin{aligned} A(2, 2) &= A(1, A(2, 1)) = A(1, A(1, A(2, 0))) = A(1, A(1, A(1, A(1, 1)))) = A(1, A(1, A(1, A(0, A(1, 0))))) = \\ &= A(1, A(1, A(1, A(0, 2)))) = A(1, A(1, A(1, 3))) = \dots \end{aligned}$$

## FUNZIONE CARATTERISTICA

Se  $B \subseteq \mathbb{N}^m$  indicheremo con  $f_B$  la funzione m-aria detta CARATTERISTICA tale che :

$$f_B(x_1, \dots, x_m) = \begin{cases} 1 & (x_1, \dots, x_m) \in B \\ 0 & \text{altrimenti} \end{cases}$$

Diremo che  $B$  è ricorso primitivo se  $f_B$  lo è.

Diremo che  $B$  è calcolabile se  $f_B$  lo è.

## PROPOSIZIONE

$B \subseteq \mathbb{N}^m$  "appartiene" a una classe PRC  $C$  se e solo se lo è anche

$$B' = \{ [x_1, \dots, x_m] \mid (x_1, \dots, x_m) \in B \}$$

## Dimostrazione

$\Rightarrow$

$$x \in B' \Leftrightarrow f_{B'}(x) = 1 \Leftrightarrow f_B((x)_1, \dots, (x)_m) = 1 \wedge \ell(x) \leq m \wedge x > 0$$

$B$  deve avere al più  $m$  divisori primi e sia maggiore di 0

Quindi se  $B$  appartiene a una classe PRC allora vi appartiene anche  $B'$

$\Leftarrow$

$$(x_1, \dots, x_m) \in B \Leftrightarrow f_B(x_1, \dots, x_m) = 1 \Leftrightarrow f_{B'}([x_1, \dots, x_m]) = 1$$

## TEOREMA

Se  $A, B \subseteq \mathbb{N}^M$  sono in una classe PRC allora anche  $A \cup B, A \cap B,$

$\bar{A} = \mathbb{N}^M \setminus A, \bar{B}$  sono nella stessa classe PRC

## DIMOSTRAZIONE

Basta osservare che:

$$f_{A \cup B} = f_A \vee f_B$$

$$f_{A \cap B} = f_A \wedge f_B$$

$$f_{\bar{A}} = \neg f_A$$

E.s.

Per trovare un insieme ricorsivo primitivo basta scegliere un predicato

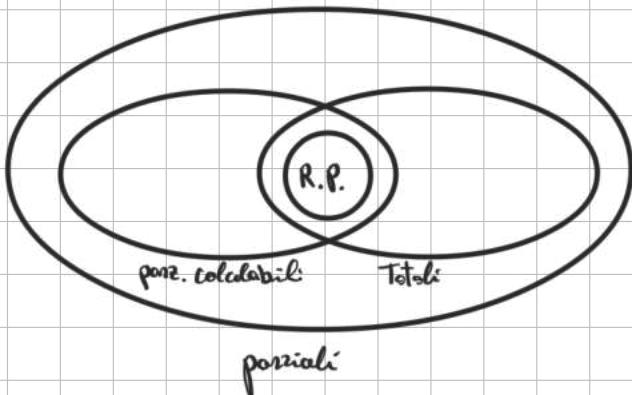
ricorsivo e considerare i valori per cui vale 1, ad esempio poiché l'insieme dei numeri primi è ric. prim., l'insieme dei numeri primi è ric. prim.

Poiché HALT non è calcolabile, l'insieme  $\{(x,y) \in \mathbb{N}^2 \mid \text{HALT}(x,y)\}$  non è ricorsivo.

Analogamente  $\{[x,y] \mid \text{HALT}(x,y)\}$  non è ricorsivo (anche considerare l' $n$ -pla applica la funzione numero di Gödel).

## SCHEMA RIASSUNTIVO

funzioni  $\mathbb{N}^m \rightarrow \mathbb{N}$



## INSIEMI RICORSIVAMENTE ENUMERABILI

Diremo che  $B \subseteq \mathbb{N}$  e' (r.e.) se

$\exists f$  funzione parz. calcolab. siffatta tale che  $B = \{x \in \mathbb{N} \mid f(x) \downarrow\}$

## PROCEDURE DI DECISIONE E SEMI-DECISIONE

In base alla tesi di Church-Turing

$B$  e' ric. prim. se esiste una procedura di **SEMI-DECISIONE** per  $B$ , cioe'  
un programma (come quello che calcola  $f$ ) che termina solo per gli input  
in  $B$ .

Se invece un dato  $B$  e' ricorsivo, vuol dire che per esso esiste  
una procedura di **DECISIONE**, cioe' un programma (come quello che calcola  
 $f_B$ ) che per ogni  $x$  e' in grado di dire in tempo finito se appartiene  
a  $B$  o no.

## TEOREMA

Se  $B \subseteq \mathbb{N}$  è ricorsivo, allora è r.e.

## DIMOSTRAZIONE

Usiamo  $f_B$  per scrivere un programma che termini solo per gli input appartenenti a  $B$

[A] IF  $\neg f_B(x)$  GOTO A

Allora  $\psi_p(x) = \begin{cases} 0 & x \in B \\ \uparrow & \end{cases}$

## TEOREMA

$B \subseteq \mathbb{N}$  è ricorsivo se e solo se  $\bar{B} \subseteq \bar{\mathbb{N}}$  sono r.e.

## DIMOSTRAZIONE

$\Rightarrow$

Se  $B$  è ricorsivo anche  $\bar{B}$  lo è e dunque sono entrambi r.e.

$\Leftarrow$

Supponiamo che esistano  $f, g$  prg. calc. tali che

$$B = \{x \in \mathbb{N} \mid f(x) \downarrow\} \quad e \quad \bar{B} = \{x \in \mathbb{N} \mid g(x) \downarrow\}$$

Sono  $p$  e  $q$  i numeri di programmi che calcolano  $f$  e  $g$

Scriviamo un programma per  $f_B$ :

[A] IF  $\text{STP}(x, p, z)$  GOTO C       $x \in B \Rightarrow y=1$

IF  $\text{STP}(x, q, z)$  GOTO E       $x \in \bar{B} \Rightarrow x \notin B \Rightarrow y=0$

$z \leftarrow z + 1$

incrementiamo il numero di passi massimi

GOTO A

ripeti il controllo

[c]  $y \leftarrow y + 1$





E.s.

②  $H_1(x) = \begin{cases} 1 & \text{if } D(x,x) \downarrow \\ \uparrow & \end{cases}$

Dimostrare che  $H_1(x)$  è parzialmente calcolabile

[A] IF  $STP(x,x,z)$  GOTO B

$z \leftarrow z+1$

GOTO A

[B]  $y \leftarrow y+1$

Ottiene potremo scrivere

$z \leftarrow D(x,x)$

Se  $D$  è definita per  $(x,x)$  passa all'istruzione  
successiva e termina. Se  $D$  non è definita  
allora andrà in loop così come il nostro  
programma.

③ dato il programma  $P$  tale che  $H_P(x_1, x_2)$  tale che il programma  
si ferma sugli input  $x_1, x_2$  non è calcolabile

Prendiamo un predicato non calcolabile ad esempio HALT.

Per "diventare" HALT consideriamo come funzione la funzione universale che ha come secondo input proprio il numero di un programma.

### TEOREMA

Se  $B, C$  sono r.e. anche  $B \cap C$  lo è

### DIMOSTRAZIONI

Essendo  $B, C$  r.e. esistono due funzioni marine parz. calcolabili  $f, g$  tali che  $B = \{x \mid f(x) \downarrow\}$  e  $C = \{x \mid g(x) \downarrow\}$  (insiemi di definizione)

Possiamo usare  $f$  e  $g$  come macro in un programma essendo parz. calcolabili.

Prendiamo un programma che termina per tutti e soli gli input

$x \in B \cap C$

$Z \leftarrow f(x)$  Il programma termina solo per i domini di  $f$  e  $g$

$Z \leftarrow g(x)$  avendo elementi di  $B \cap C$ , altrimenti va in loop

## TEOREMA

Se  $B, C$  sono r.e. anche  $B \cup C$  lo è

## DIMOSTRAZIONI

Essendo  $B, C$  r.e. esistono due funzioni umane ponz. calc.  $f, g$

tali che  $B = \{x \mid f(x) \in \}\} \cup C = \{x \mid g(x) \in \}$  (insiemi di definizione)

Sono  $p, q$  rispettivamente numeri di programmi che calcolano  $f$  e  $g$

Prendiamo un programma che termina per tutti e soli gli input

$x \in B \cup C$ , si attesta il predicato STP

[A] IF  $STP(x, p, z)$  GOTO E

IF  $STP(x, q, z)$  GOTO E

$z \leftarrow z + 1$

GOTO A

vediamo se  $p$  termina in  $Z$  posso con l'input  $X$ , quindi  $X$  appartiene al dominio di  $p$  e di conseguenza all'unione

Il programma terminerà solo se  $x$  appartiene a  $B \cup C$

## OSS

Nella prima dimostrazione non poteremo usare STP perché se avessimo preso come primo input un  $x \in C$  ma non a  $B$  il programma sarebbe comunque terminato

Generalmente quando vogliamo verificare il dominio di almeno un insieme possiamo usare STP che non va in loop se non verificato.

### INSIEME $W_m$

Definiamo per  $m \in \mathbb{N}$

$$W_m = \left\{ x \in \mathbb{N} \mid \overline{\Phi}(x, m) \downarrow \right\} = \left\{ x \in \mathbb{N} \mid \Psi_{P(m)}^{(1)}(x) \downarrow \right\}$$

$W_m$  è il dominio della funzione calcolata dal programma numero  $m$

### TEOREMA DI ENUMERAZIONE

$$B \text{ r.e.} \Leftrightarrow \exists m \mid B = W_m$$

### DIMOSTRAZIONE

$$B \text{ r.e.} \Leftrightarrow \exists f \text{ porz. calc.} \mid B = \left\{ x \mid f(x) \downarrow \right\} \Leftrightarrow \exists m \mid f = \Psi_{P(m)}^{(1)} \text{ e}$$

$$B = \left\{ x \mid f(x) \downarrow \right\} \Leftrightarrow \exists m \mid B = W_m$$

### INSIEME DIAGONALE

$$K = \left\{ m \in \mathbb{N} \mid m \in W_m \right\} = \left\{ m \mid \overline{\Phi}(m, m) \downarrow \right\}$$

### OSS

$$\text{Possiamo dire che } m \in W_m \Leftrightarrow \overline{\Phi}(m, m) \downarrow \Leftrightarrow \text{HALT}(m, m) \Leftrightarrow \exists t \mid \text{STP}(m, m, t)$$

## PROPOSIZIONE

$K$  è r.e. ma non riconoscibile

## DIMOSTRAZIONE

$K$  è r.e. in quanto dominio della funzione  $H_1$ .

Possiamo dimostrare che non è riconoscibile in 2 modi:

①  $K$  è riconoscibile  $\Leftrightarrow \bar{K}$  è riconoscibile (insieme ANTI DIAGONALE)

Se per assurdo  $\bar{K}$  fosse r.e. allora dovrebbe esistere un  $i$  tale che  $\bar{K} = W_i$ .

Ma allora  $i \in \bar{K} \Leftrightarrow i \in W_i \Leftrightarrow i \in K$  assurdo.

② La funzione caratteristica  $f_K$  verifica:

$$f_K(x) = \begin{cases} 1 & x \in K \quad (\Leftrightarrow \text{HALT}(x,x)) \\ 0 & \end{cases}$$

cioè  $f_K(x) = \text{HALT}(x,x)$  che non è calcolabile.

## PROPOSIZIONE

Se  $B$  è r.e.,  $\exists R$  predicato binario ric. |  $B = \{x \in \mathbb{N} \mid (\exists t) R(x,t)\}$

## DIMOSTRAZIONE

Se  $B$  è r.e. sia  $B = W_m$  allora  $B = \{x \in \mathbb{N} \mid (\exists t) \text{STP}(x,m,t)\}$

dato che  $W_m = \{x \in \mathbb{N} \mid \text{HALT}(x,m)\}$

## TEOREMA

Sia  $B \neq \emptyset$  e s.e. allora  $\exists f$  ric. prim.  $| B = \{f(m) \mid m \in \mathbb{N}\} \}$

## DI MOSTRAZIONE

Per quanto scritto sopra  $\exists R$  predicato ric. prim.  $| B = \{x \mid (\exists t) R(x, t)\}$

Sia  $x_0 \in B$ , definiamo

$$f(x) = \begin{cases} l(x) & R(l(x), r(x)) \\ x_0 & \text{altrimenti} \end{cases}$$

Se  $m \in B$ ,  $\exists t \mid R(m, t)$ . Posto  $x = \langle m, t \rangle$  si ha:

$$f(x) = f(\langle m, t \rangle) = m$$

Inoltre  $\forall n \in \mathbb{N}, f(n) \in B$ , infatti  $f(n) = l(n)$  se  $R(l(n), r(n))$  ma

dato che  $B = \{m \mid (\exists t) R(m, t)\}$  in tal caso si ha  $l(n) \in B$

(per  $n = l(n) \exists t = r(n) \mid R(n, t)$ )

Abbiamo quindi dimostrato che  $B$  appartiene all'insieme delle immagini e viceversa

## TEOREMA

Sia  $f$  unaria e parz. calc. e  $B = \{f(m) \mid f(n) \downarrow\}$  allora  $B$  è s.e.

## DIMOSTRAZIONE

Sia  $p$  il numero di un programma che calcola  $f$ .

Scriviamo un programma che termina se  $X=f(m)$  per qualche  $m$  (tale che  $f(m) \downarrow$ ):

[A] IF  $\neg \text{STP}(z_1, p, z_2)$  GOTO B

se  $f$  non è definita in  $z_1$   
andrai in loop

$Z_3 \leftarrow f(z_1)$

IF  $Z_3 = X$  GOTO E

$x$  è calcolato da  $f$

[B]  $Z_1 \leftarrow Z_1 + 1$

proviamo a verificare un altro input

IF  $Z_1 \leq Z_2$  GOTO A

se non abbiamo raggiunto il max di possibili

$Z_2 \leftarrow Z_2 + 1$

aumentiamo i possibili massimi

$Z_1 \leftarrow 0$

e facciamo ripartire gli input

GOTO A

Infatti se  $m$  è tale che  $f(m) \downarrow$  e  $x=f(m) \downarrow$ , esiste un valore  $Z_2$

tale che  $\text{STP}(m, p, Z_2)$ , e il programma prima o poi raggiungerà

$Z_1 = m$  e  $Z_2 \geq t$ .

L'unica condizione che chiude la terminazione del programma è nella terza istruzione, cioè quando si trova un tale  $m$

## TEOREMA

Sia  $B \subseteq \mathbb{N} \setminus \emptyset$ . Sono equivalenti:

- ①  $B$  r.e. cioè  $\exists g$  part. calc. tale che  $B = \{x \in \mathbb{N} \mid g(x) \downarrow\}$
- ②  $B = \{f(m) \mid m \in \mathbb{N}\}$  per qualche  $f$  ricorsiva primitiva
- ③  $B = \{f(m) \mid m \in \mathbb{N}\}$  per qualche  $f$  calcolabile
- ④  $B = \{f(m) \mid f(m) \downarrow\}$  per qualche  $f$  part. calc.

## DI MOSTRAZIONE

$1 \Rightarrow 2$  è già dimostrato;  $2 \Rightarrow 3 \Rightarrow 4$  ovvio

## LINGUAGGIO S E ALTRI STRUMENTI DI CALCOLO

Abbiamo visto come il linguaggio S usa solo funzioni molto elementari, tuttavia alcune di queste risultano tali se usiamo una notazione "umana" come ad esempio le operazioni di incremento e decremento. Per questo motivo passiamo ad altri linguaggi le operazioni elementari saranno sostituite da altre operazioni per rappresentazioni diverse (base 2, 16 etc...) ottenendo linguaggi  $S_n$  equivalenti.

## MACCHINE DI TURING

Una macchina di Turing è formata da una testina che scorre su un nastro infinito che in ogni posizione ha un simbolo che possiamo essere sia una cifra (il nastro rappresenta l'odierna memoria).

La macchina esegue delle istruzioni come lettura e scrittura dal nastro e calcola funzioni partendo da uno stato iniziale e da una rappresentazione di input sul nastro; l'output corrispondente sarà scritto sul nastro al termine del calcolo (se finita)

Formalmente, una macchina di Turing è data da:

$(Q, q_1, A, I)$  con

- $Q$  insieme finito (di STATI)
- $q_1 \in Q$  stato iniziale
- $A$  insieme finito (ALFABETO)
- $I \subseteq Q \times (A \cup \{\#\}) \times (A \cup \{L, R\}) \times Q$

L e R indicano la direzione di movimento della testa

### CALCOLO DI UNA MACCHINA DI T.

Il calcolo di una macchina di Turing si svolge:

Ad ogni istante la macchina è in uno stato  $q \in Q$  e legge

un simbolo  $a \in A$  sul nastro (oppure  $\#$ )

I primi due componenti indicano quando eseguire l'istruzione

(quando sei in quelllo stato e leggi quel simbolo). Il

terzo indica cosa fare e il quarto in che stato passare.

Dopo aver letto il simbolo  $a$  verifica se tra le istruzioni in  $I$

c'è un'che inizia con tale coppia stato-simbolo (controllabile

tutte una ad una) e se ne trova una valida la "esegue" cioè

se ad esempio  $(q, a, b, q') \in I$  allora sostituisce nella

posizione corrente sul nastro a con b se  $b \in A \cup \{\#\}$  oppure in alternativa si muove a sinistra (o destra) se  $b \in \{L, R\}$ ; in ogni caso poi passa allo stato  $q'$ .

Se non viene trovata alcuna quadrupla " valida" la macchina si ferma.

OSS

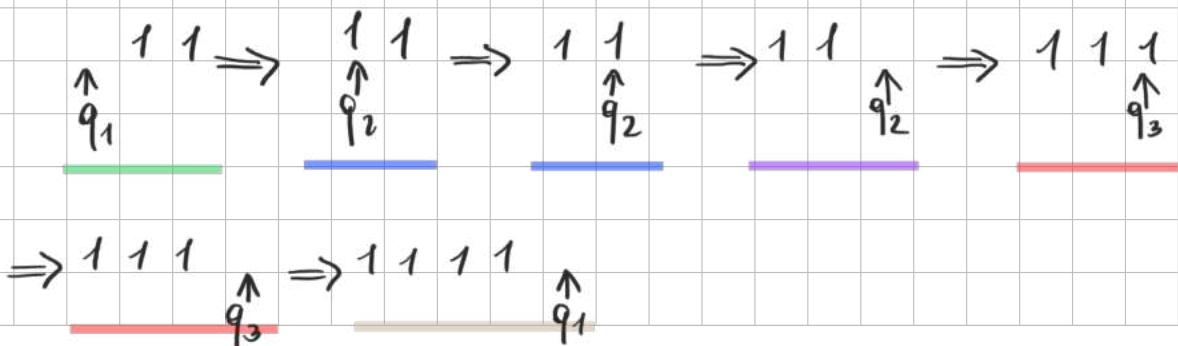
Se ci sono più quadruple eseguibili parleremo di macchine **NON DETERMINISTICHE** ed esegue la prima che trova. Altrimenti parleremo di macchine deterministiche

E.s.

$$m = (Q, q_1, A, I) \text{ con } Q = \{q_1, q_2, q_3\}, A = \{1\},$$

$$I = \{(q_1, \leftarrow, R, q_2), (q_2, 1, R, q_2), (q_2, \leftarrow, 1, q_3), (q_3, 1, R, q_3), (q_3, \leftarrow, 1, q_1)\}$$

Vediamo il calcolo con input (nastro) 2-11 (base 1)



L'istruzione calcola  $f(x) = x+2$

### INSIEME RICORSIVAMENTE ENUMERABILE

Per l'equivalenza tra linguaggio  $S$  e macchina di  $T$ , un insieme

$A = \{a_1, \dots, a_m\}$  è **RICORSIVAMENTE ENUMERABILE** se e solo se

è l'insieme degli input sui quali il calcolo di una m.d.T.

termina (li chiamiamo input **ACCETTATI**)

OSS

- Tutto ciò vale nel caso di alfabeti  $A$  qualsiasi: possiamo parlare di stringhe di input accettate che costituiscono il linguaggio accettato.
- L'insieme di tutte le stringhe (o parole) su  $A$  (inclusa quella vuota  $\epsilon$ ) si denota con  $A^*$ . Un **LINGUAGGIO** è un sottinsieme di  $A^*$ .

### LINGUAGGIO RICORSIVAMENTE ENUMERABILE

Un linguaggio è **RICORSIVAMENTE ENUMERABILE** se è linguaggio accettato da una m.d.T.

## AUTOMA A STATI FINITI DETERMINISTICO (DFA)

E' un MODELLO DI CALCOLO SU STRINGHE costituito da una 5-pla  
 $m = (Q, A, \delta, q_0, F)$  dove

- $Q$  insieme finito (di STATI)
- $A$  insieme finito (ALFABETO)
- $q_0 \in Q$  stato iniziale
- $F \subseteq Q$  e' l'insieme di stati TERMINALI (o di accettazione) di  $m$
- $\delta$  e' una funzione totale  $Q \times A \rightarrow Q$  detta di TRANSIZIONE

### OSS

Le DFA possono essere viste come macchine di T. che si muovono  
sempre verso destra senza scivolare sul nastro

## FUNZIONE $\delta^*$

Per un tale automa  $m$  definiamo

$$\delta^*: Q \times A^* \rightarrow Q \quad \text{che}$$

$$\forall q \in Q \quad \delta^*(q, \varepsilon) = q \quad (\text{parola vuota, non cambio stato})$$

$$\forall q \in Q, u \in A^*, a \in A, \quad \delta^*(q, ua) = \delta(\delta^*(q, u), a)$$

$\delta$  e' definita su una lettera,  $\delta^*$  sulle parole

## LINGUAGGIO ACCETTATO

Chiamiamo **LINGUAGGIO ACCETTATO** da  $m$  l'insieme

$$L(m) = \{w \in A^* \mid S^*(q_1, w) \in F\}$$

Diremo che  $L \subseteq A^*$  è **REGOLARE** se esiste una DFA su quell'alfabeto  $A$  tale che  $L = L(m)$

### OSS

Per prodotto di stringhe si intende la concatenazione di queste, analogamente vale con le potenze

E.s.

Sull'alfabeto  $\{a, b\}$  il linguaggio  $L = \{a^n b^m \mid n, m > 0\}$  è regolare perché accettato dal seguente DFA:

$$Q = \{q_1, q_2, q_3, q_4\}, F = \{q_4\}$$

$S$	a	b
$q_1$	$q_2$	$q_3$
$q_2$	$q_2$	$q_4$
$q_3$	$q_3$	$q_3$
$q_4$	$q_3$	$q_4$

Nello stato  $q_1$ , se leggo  $b$  vado in  $q_3$ . In  $L$  non ci sono parole che iniziano con "b", quindi in  $q_3$  restiamo in  $q_3$  sempre. È detto stato **POZZO**

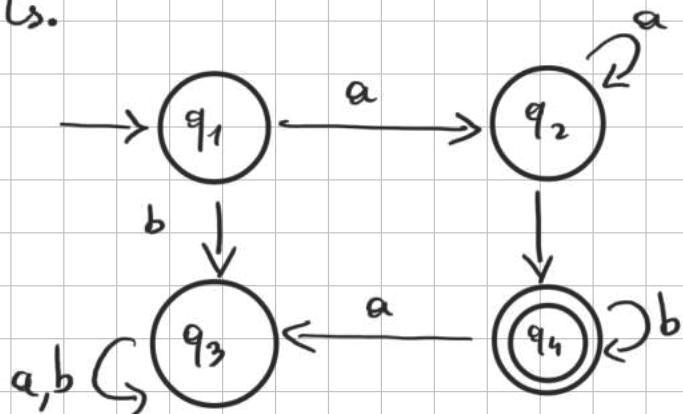


## DIAGRAMMA DI TRANSIZIONE

Indichiamo con delle "bolle" gli stati e con delle frecce le transizioni etichettate con la lettera che stiamo leggendo.

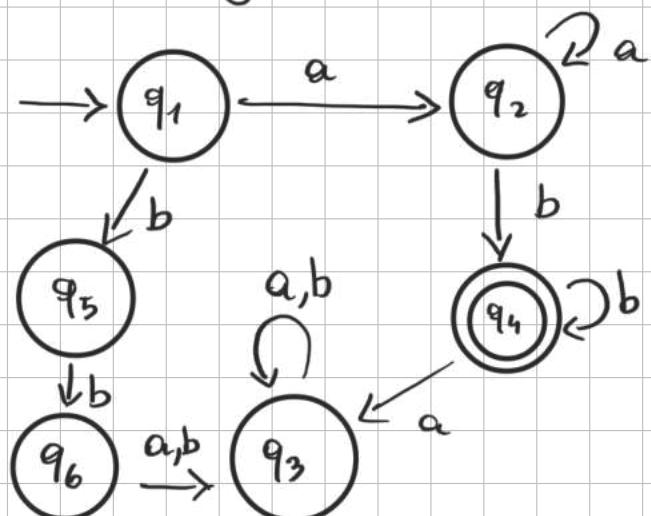
E' buona norma segnalare lo stato iniziale e lo stato finale, quest'ultimo contrassegnato dal doppio bordo.

E.s.



$q_3$  è lo stato porro

Analizziamo il caso analogo con  $L' = L \cup \{bb\}$



## OSS

Alcune notazioni diverse sono

- $\emptyset$  per la stringa vuota (notch'  $\epsilon$ )
- $w^{[n]}$  per la ripetizione  $ww...w$  n volte (notch'  $w^n$ )

## PROPOSIZIONE

Sia  $L \subseteq A^*$  e sia  $A \subseteq A'$ , allora esiste un DFA su  $A$  che accetta  $L$  se e solo se ne esiste uno su  $A'$

## DIMOSTRAZIONE

=>

Sia  $m = (Q, A, \delta, q_1, F)$  un DFA tale che  $L(m) = L$ .

Per costruire un DFA  $m'$  su  $A'$  tale che  $L(m') = L$  basta

aggiungere uno stato porro:

$$m' = (Q \cup \{q_p\}, A', \delta', q_1, F) \text{ con}$$

$$\delta' = \begin{cases} \delta(q, a) & \text{se } q \in Q \wedge a \in A \\ q_p & \text{altrimenti} \end{cases}$$

$\Leftarrow$

Sia  $m' = \{Q', A', \delta', q_1', f'\}$  un DFA su  $A'$  che accetti  $L$ .

Allora "ristringiamo"  $\delta'$  alle sole letture valide:

$$m = (Q', A, \delta = \delta' |_{\{a \in A\}}, q_1', f')$$

E' un DFA su  $A$  che accetta lo stesso linguaggio  $L(m) = L$  poiché le parole di  $L$  contengono solo lettere di  $A$ .

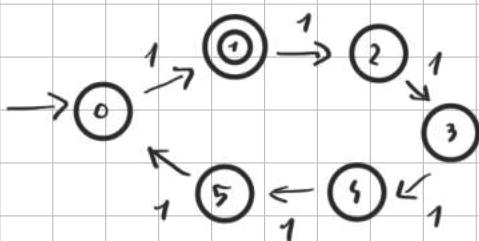
### Esercizi di SINTESI

- Dimostrare che  $L_1 = \{1^{6k} \cdot 1 \mid k \geq 0\}$  e' regolare

Consideriamo un alfabeto solo di 1.

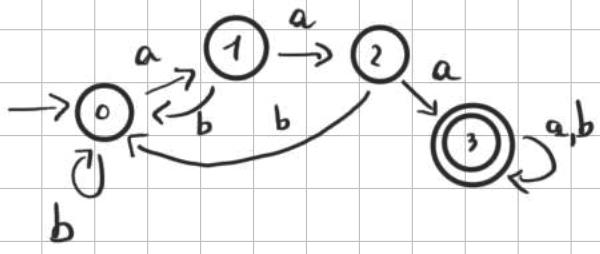
Ogni parola e' composta da  $6k+1$  1 quindi  $L = \{1^{6k+1} \mid k \geq 0\}$

Disegniamo l'automa



Si noti che non vi e' alcuno stato pazzo

• Dimostrare che  $L_1 = \{w \in A^* \mid w \text{ contiene 3 "a" consecutive}\}$  è regolare

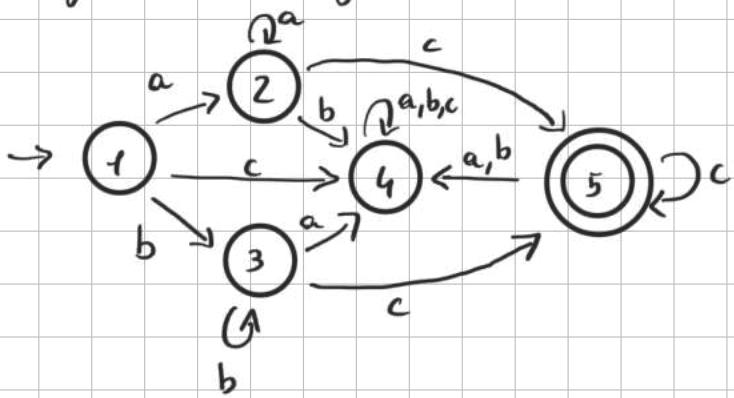


Esercizi di ANALISI

S	a	b	c
q <sub>1</sub>	q <sub>2</sub>	q <sub>3</sub>	q <sub>4</sub>
q <sub>2</sub>	q <sub>2</sub>	q <sub>4</sub>	q <sub>5</sub>
q <sub>3</sub>	q <sub>4</sub>	q <sub>3</sub>	q <sub>5</sub>
q <sub>4</sub>	q <sub>4</sub>	q <sub>4</sub>	q <sub>4</sub>
q <sub>5</sub>	q <sub>4</sub>	q <sub>4</sub>	q <sub>5</sub>

$F = \{q_5\}$

Disegniamo il diagramma di transizione



OSS

La parola vuota è accettata se e solo se lo stato iniziale  
è di accettazione. Formalmente

$$m = (Q, A, \delta, q_1, F), \quad \varepsilon \in L(m) \Leftrightarrow q_1 \in F$$

## AUTOMI FINITI NON DETERMINISTICI NFA

Un NFA è una 5-plo

$$m = (Q, A, \delta, q_1, F)$$

- $Q$  insieme finito (di **STATI**)
- $A$  insieme finito (**ALFABETO**)
- $q_1 \in Q$  stato iniziale
- $F \subseteq Q$  è l'insieme di stati **TERMINALI** (o di accettazione) di  $m$
- $\delta: Q \times A \rightarrow P(Q)$  funzione di **TRANSIZIONE** ( $P$  insieme delle parti)

Un NFA accetta  $w \in A^*$  se leggendo a partire dello stato iniziale arriva a un insieme di stati che contenga almeno uno terminale.

Formalmente definiamo

$$\delta^*: Q \times A^* \rightarrow P(Q)$$

in modo che  $\forall q \in Q \quad \delta^*(q, \varepsilon) = \{q\}$  e

$$\forall u \in A^*, a \in A \quad \delta^*(q, ua) = \bigcup_{q' \in \delta^*(q, u)} \delta(q', a) \quad \text{da } q' \text{ leggono } a$$

$\underbrace{\hspace{10em}}$

insieme degli stati possibili dopo aver letto "u"

## LINGUAGGIO ACCETTATO NFA

Dato una NFA  $m$ , definiamo il linguaggio accettato su  $m$  come

$$L(m) = \{w \in A^* \mid S^*(q_1, w) \cap F \neq \emptyset\}$$

## PREPOSIZIONE

Se  $L \subseteq A^*$  è regolare, esiste un NFA  $m'$  tale che  $L(m') = L$

## Dimostrazione

Dato  $m = (Q, A, S, q_1, F)$  che accetta  $L$ , allora  $m' = (Q, A, S', q_1, F)$  con

$S' = (q, a) = \{S(q, a)\} \quad \forall q \in Q \text{ e } \forall a \in A$  è un NFA tale che

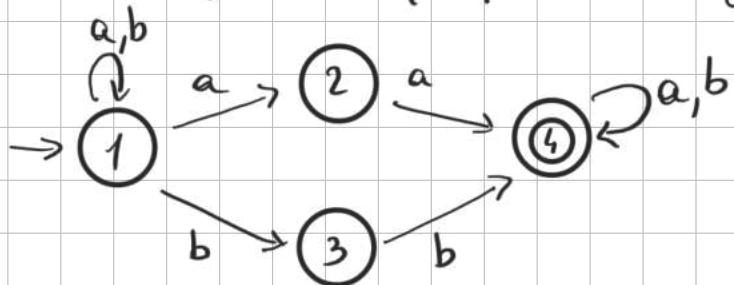
$$L(m') = L(m) = L$$

## OSS

Il diagramma di transizione di un NFA si disegna analogamente al DFA ma da ogni stato potranno uscire più frecce etichettate con la stessa lettera (o nessuna).

E.

Disegnare il diagramma di transizione di un NFA che accetti  
tutte le parole in  $\{a,b\}$  che contengono 2 "a" o 2 "b" consecutive



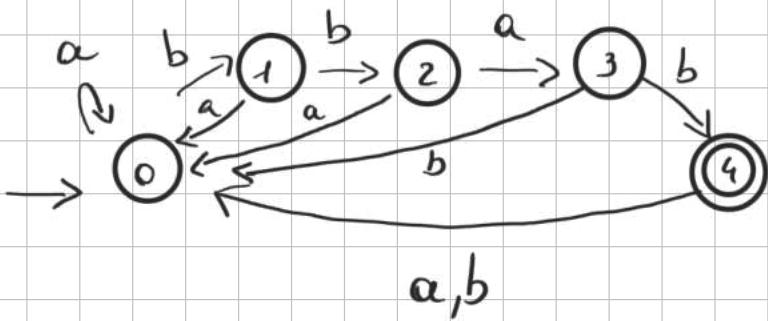
$$S^*(q_1, ab) = \{q_1\}$$

pj 241 n° 1 b,g,i, 3 b,c, 4,5,8

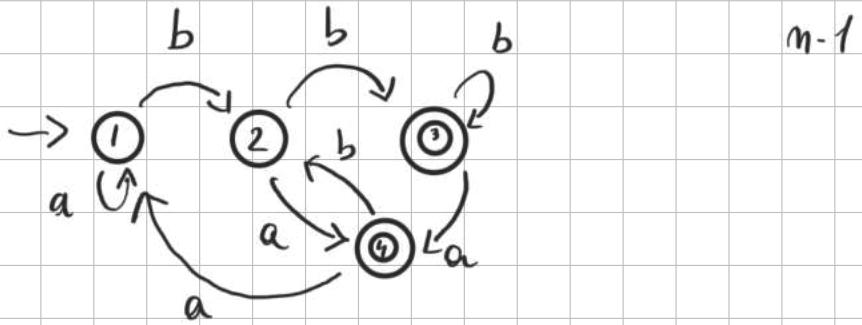
pj 246 n° 1

1. In each of the following examples, an alphabet  $A$  and a language  $L$  are indicated with  $L \subseteq A^*$ . In each case show that  $L$  is regular by constructing a finite automaton  $\mathcal{M}$  that accepts  $L$ .

(b)  $A = \{a, b\}$ ;  $L$  consists of all words whose final four symbols form the string  $bbab$ .

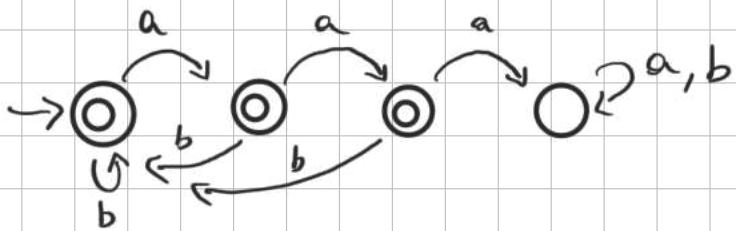


- (g)  $A = \{a, b\}$ ;  $L$  consists of all strings  $s_1 s_2 \dots s_n$  such that  $s_{n-2} = b$ .  
 (Note that  $L$  contains no strings of length less than 3.)



$m-1$

- (i)  $A = \{a, b\}$ ;  $L$  consists of all words in which three  $a$ 's do not occur consecutively.

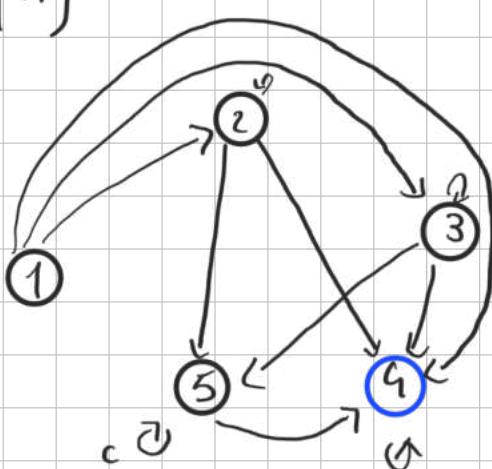


3. Describe the language accepted by each of the following finite automata. In each case the initial state is  $q_1$ .

b)

$\delta_1$	$a$	$b$	$c$
$q_1$	$q_2$	$q_3$	$q_4$
$q_2$	$q_2$	$q_4$	$q_5$
$q_3$	$q_4$	$q_3$	$q_5$
$q_4$	$q_4$	$q_4$	$q_4$
$q_5$	$q_4$	$q_4$	$q_5$

$$F_2 = \{q_4\}$$

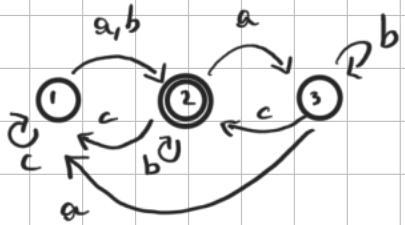


$x \stackrel{a}{\rightarrow} y \stackrel{b}{\rightarrow} z \stackrel{w}{\rightarrow} k$   
 $c \stackrel{b}{\rightarrow} a \stackrel{b}{\rightarrow} c$

c)

$\delta_3$	a	b	c
$q_1$	$q_2$	$q_2$	$q_1$
$q_2$	$q_3$	$q_2$	$q_1$
$q_3$	$q_1$	$q_3$	$q_2$

$$F_3 = \{q_2\}.$$



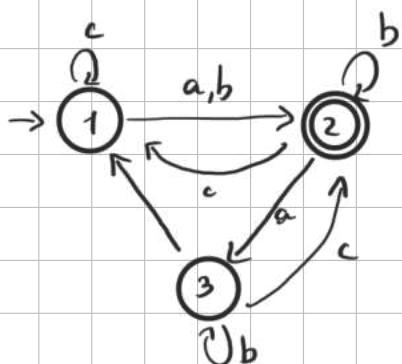
- 8.\* Let  $\mathcal{M}$  be a finite automaton on the alphabet  $A = \{s_1, \dots, s_n\}$  with states  $Q = \{q_1, \dots, q_m\}$ , transition function  $\delta$ , initial state  $q_1$ , and accepting states  $F$ . Give a Turing machine  $\mathcal{M}'$  that accepts  $L(\mathcal{M})$ .

$$\mathcal{M}' = (Q', q_1, A, T)$$

$$Q' =$$

## CORREZIONE ESERCIZI

3c)



$$c^n \times b^m \mid x \in \{a, b\}$$

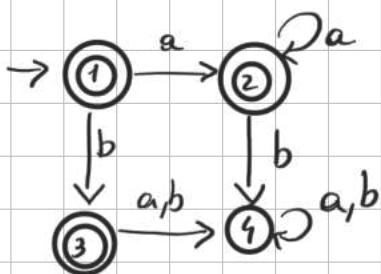
parola per arrivare da  $q_1$  a  $q_2$

I modi di andare da  $q_1$  a  $q_1$  senza passare nuovamente per  $q_1$

Sono:

$$\left( (\{c\} \cup \{a, b\}) \cdot (\{b, ab^m\})^{m_2} \cdot \{c, ab^{m_3}a\} \right)^{m_4} \cdot \{a, b\} \cdot (\{b, ab^{m_5}c\})^{m_6}$$

$$5) L = \{a^n \mid n \geq 0\} \cup \{b\}$$



Questo linguaggio non puo' essere accettato da nessun automa

deterministico con un solo stato d'accettazione perché lo stato iniziale dev'essere d'accettazione ma per come è definito avremo

sempre almeno altri due stati d'accettazione ovvero quello

8) Supponiamo che  $L$  sia accettato da una NFA

Sia  $L = L(m)$  con  $m = (Q, A, \delta, q_1, F)$

Allora  $m' = (Q \cup \{q_0\}, q_0, A, I)$  con

$$I = \{(q_0, \sqcup, R, q_1)\} \cup \{q_1, a, R, \delta(q_1, a) \mid q_1 \in Q \wedge a \in A\} \cup$$

$\{(q_1, \sqcup, R, q) \mid q \in Q \setminus F\}$  e' una MdiT tale che  $L(m') = L$

### TEOREMA

Se  $L \subseteq A^*$  e' regolare, anche  $\bar{L} = A^* - L$  e' regolare

### DIMOSTRAZIONE

Sia  $m = (Q, A, \delta, q_1, F)$  un DFA tale che  $L(m) = L$

Allora  $\bar{m} = (Q, A, \delta, q_1, Q \setminus F)$  accetta  $\bar{L}$

### COROLLARIO

Se  $L$  e' regolare allora e' ricorsivo

### TEOREMA

$L$  e' regolare  $\Leftrightarrow L = L(m)$  per qualche NFA  $m$

## DIMOSTRAZIONE

$\Rightarrow$

Sia  $m = (Q, A, \delta, q_1, F)$  un DFA tale che  $L = L(m)$

$m' = (Q, A, \delta', q_1, F)$  con  $\delta'(q, a) = \{\delta(q, a)\} \quad \forall q \in Q, a \in A$

$m'$  e' un NFA che accetta  $L$  (banalmente)

$\Leftarrow$

Sia  $m = (Q, A, \delta, q_1, F)$  un NFA tale che  $L(m) = L$

Definiamo  $m' = (\wp(Q), A, \delta', \{q_1\}, F')$  con

$F' = \{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\}$  e  $\delta'(Q', a) = \bigcup_{q \in Q'} \delta(q, a)$

Si puo' dimostrare che  $\forall w \in A^*, Q' \subseteq Q \quad \delta'^*(Q', a) = \bigcup_{q \in Q'} \delta^*(q, w)$

Allora  $L(m') = \{w \in A^* \mid \delta'^*(\{q_1\}, w) \in F'\} = \{w \in A^* \mid \delta^*(q_1, w) \in F'\} =$

$= \{w \in A^* \mid \delta^*(q_1, w) \cap F \neq \emptyset\} = L(m)$

## DFA NONRESTARTING

Un DFA e' NONRESTARTING se  $m = (Q, A, \delta, q_1, F)$  e  $q_1 \in \delta(Q \times A)$

(non e' possibile tornare allo stato iniziale)

## PROPOSIZIONE

Se  $L$  è regolare allora esiste un DFA non restarting che lo accetta

## DIMOSTRAZIONE

Sea  $m = (Q, A, S, q_1, F)$  un DFA tale che  $L = L(m)$

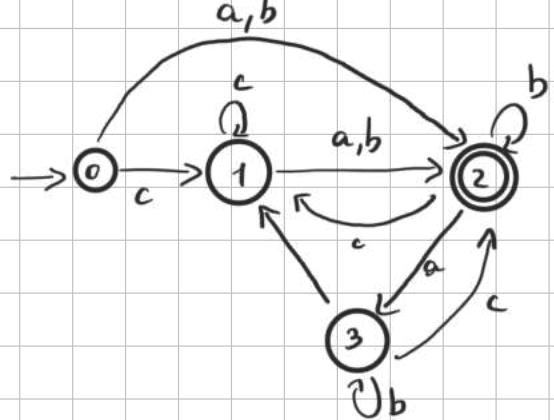
Sia  $q_0 \in Q$  e definiammo  $m' = \{Q \cup \{q_0\}, A, S', q_0, F'\}$  dove

$$S'(q, a) = S(q, a) \quad \forall a \in A \quad e$$

$$F' = \begin{cases} F & se q_1 \in F \\ F \cup \{q_0\} & altrimenti \end{cases}$$

Evidentemente  $m'$  è un DFA non restarting che accetta  $L$

E.s.



Accetta lo stesso linguaggio definito prima ma è non restarting

## TEOREMA

Se  $L, L' \subseteq A^*$  sono regolari allora  $L \cup L'$  è regolare

## DIMOSTRAZIONE

Siano  $m = (Q, A, \delta, q_0, F)$  e  $m' = (Q', A, \delta', q'_0, F')$  due DFA

non starting tali che  $L = L(m)$  e  $L' = L(m')$  e  $Q \cap Q' = \emptyset$

Definiamo un NFA  $\hat{m} = (Q \cup Q' \cup \{q_0\} \setminus \{q_0'\}, A, \hat{\delta}, q_0, \hat{F})$  con  $q_0 \notin Q \cup Q'$

$$\text{e } \hat{\delta}(q, a) = \begin{cases} \{\delta(q, a)\} & \text{se } q \in Q \setminus \{q_0\} \\ \{\delta'(q, a)\} & \text{se } q \in Q' \setminus \{q_0'\} \\ \{\delta(q_0, a), \delta'(q_0, a)\} & \text{se } q = q_0 \end{cases}$$

$$\text{e } \hat{F} = \begin{cases} F \cup F' & \text{se } q_0 \in F \wedge q_0' \in F' \\ F' \cup \{q_0\} & \text{altrimenti} \end{cases}$$

Fondamentalmente prendiamo due automi non starting e li "uniamo" nello stato iniziale, si vede che  $\hat{m}$  accetta  $L \cup L'$

## COROLLARIO

Se  $L, L'$  sono regolari  $\Rightarrow L \cap L'$  regolare

## DIMOSTRAZIONE

Per De Morgan  $L \cap L' = A^* \setminus ((A^* \setminus L) \cup (A^* \setminus L'))$

## PROPOSIZIONE

$\emptyset$  è regolare

## DI MOSTRAZIONE

Basta considerare qualunque DFA con  $F \neq \emptyset$

## PROPOSIZIONE

$\forall w \in A^*, \{w\}$  è regolare

## DI MOSTRAZIONE

Sia  $w = a_1, \dots, a_m$  con  $a_i \in A$  per  $1 \leq i \leq m$

Allora l'NFA il cui diagramma è

$\rightarrow \circlearrowleft \xrightarrow{a_1} \circlearrowright \xrightarrow{a_2} \dots \xrightarrow{a_m} \circlearrowright \text{ accetta solo } w$

L'automa è  $m = (Q, A, \delta, q_0, F)$  con  $Q = \{q_0, \dots, q_m\}$ ,  $F = \{q_m\}$  e

$$\delta(q_i, a) = \begin{cases} \{q_{i+1}\} & \text{se } a = a_{i+1} \\ \emptyset & \text{altrimenti} \end{cases}$$

## COROLLARIO

Tutti i linguaggi finiti sono regolari

## PRODOTTO TRA LINGUAGGI

Dati  $L, L' \subseteq A^*$  definiamo

$$LL' = \{uv \in A^* \mid u \in L \wedge v \in L'\} \quad \text{il PRODOTTO tra } L \text{ e } L'$$

## TEOREMA

Se  $L, L' \subseteq A^*$  sono regolari lo è anche  $LL'$

## Dimostrazione

Siano  $m = (Q, A, \delta, q_1, F)$  e  $m' = (Q', A', \delta', q'_1, F')$  due DFA tali che

$$Q \cap Q' = \emptyset \quad \text{e} \quad L(m) = L, \quad L(m') = L'$$

Allora l' NFA  $\hat{m} = (Q \cup Q', A, \hat{\delta}, \{q_1\}, \hat{F})$  con  $\hat{F} = \begin{cases} F & q'_1 \notin F \\ F \cup F' & q'_1 \in F' \end{cases}$

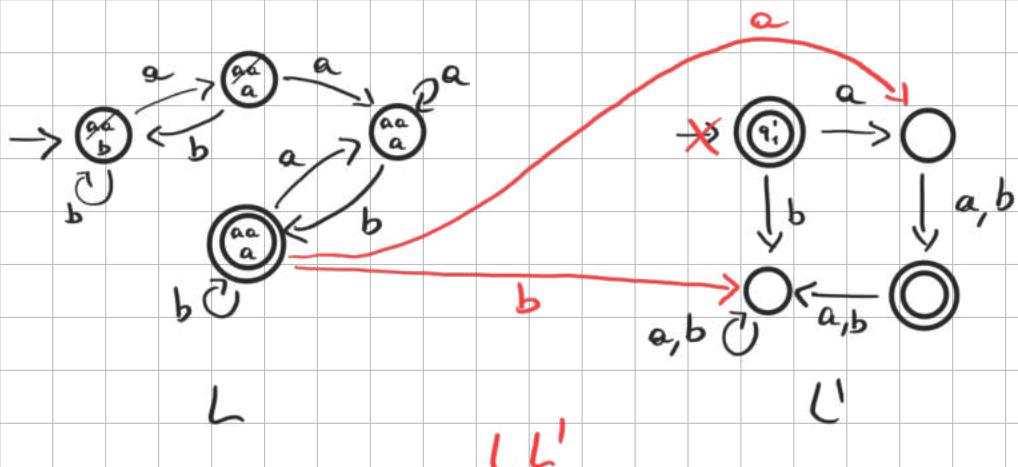
$$\hat{\delta}(q, a) = \begin{cases} \{\delta(q, a)\} & q \in Q \setminus F \\ \{\delta'(q, a)\} & q \in Q' \\ \{\delta(q, a), \delta'(q, a)\} & q \in Q \cap F \end{cases}$$

accetta  $LL'$

E.

Sia  $A = \{a, b\}$  e  $L = \{w \in A^* \mid w \text{ contiene } aa \text{ e finisce per } b\}$ ,

$$L' = \{\epsilon, aa, ab\}$$



$q_1'$  non è accessibile in  $LL'$

$L^*$

Dato  $L \subseteq A^*$  definiamo

$$L^* = \{u_1 u_2 \dots u_n \in A^* \mid n \geq 0 \text{ e } u_i \in L \text{ per } 1 \leq i \leq n\}$$

$L^*$  è l'insieme delle concatenazioni di una o più parole di  $L$

OSS

Possiamo definire  $L^n$  nel seguente modo:

- $L^0 = \{\varepsilon\}$  ed è neutro per il prodotto tra linguaggi
- $L^{n+1} = L^n \cdot L$  per  $n \geq 0$

Allora si ha  $L^* = \bigcup_{n=0}^{\infty} L^n$

## TEOREMA

$L \subseteq A^* \Rightarrow L^*$  regolare

## DIMOSTRAZIONE

Sia  $m = (Q, A, \delta, q_1, F)$  un DFA nonstarting che accetta  $L$

Sia l' NFA  $\hat{m} = (Q, A, \hat{\delta}, q_1, \{q_1\})$  con

$$\hat{\delta}(q, a) = \begin{cases} \{\delta(q, a)\} & \delta(q, a) \in F \\ \{\delta(q, a), q_1\} & \text{altrimenti} \end{cases}$$

Dimostriamo che  $\hat{m}$  accetta concatenazioni di  $m$  parole di  $L$

per induzione su  $m$ .

•  $m=0$  è caso base, banale. Supponiamo che  $\hat{m}$  accetti concatenazioni

di  $m-1$  parole.

• Sappiamo che  $L^m = L^{m-1} \cdot L$ , quindi è banalmente dimostrato per concatenazione dunque  $L^* \subseteq L(\hat{m})$ .

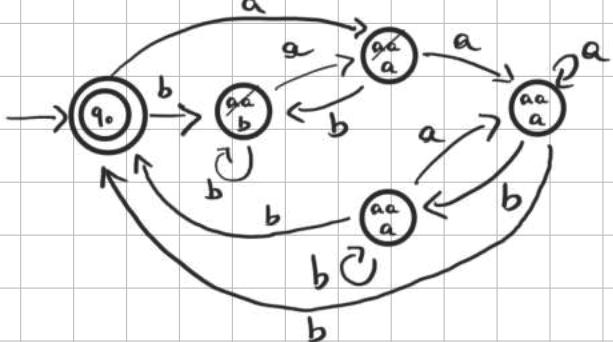
D'altronde essendo  $m$  nonstarting, l'unico modo di tornare a  $q_1$  in  $\hat{m}$  è leggere parole di  $L$ , quindi  $L(\hat{m}) \subseteq L^*$

Per doppia inclusione  $L^* = L(\hat{m})$

E<sub>s</sub>

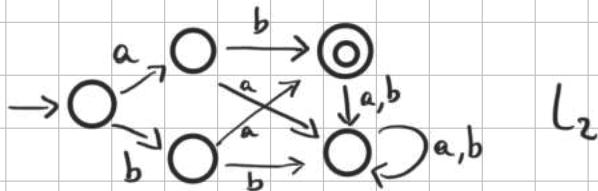
- Sia  $L = \{w \in A^* \mid w \text{ contiene } aa \text{ e finisce per } b\}$

Dobbiamo rendere l'automa nonrestaring

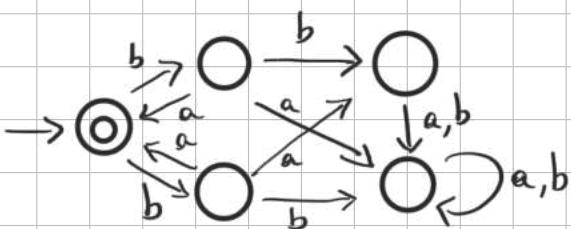


Questo automa accetta le parole di  $L^*$  che in questo caso  
e' formalmente  $L \cup \{\epsilon\}$

- Sia  $L_2 = \{ab, ba\}$ , costruiamo l'NFA che accetti  $L_2^*$  partendo  
dal DFA che accetti  $L_2$



L'automa già è nonrestaring ed ha un unico stato  
d'accettazione, completiamo l'"adattamento"



## TEOREMA DI KLEENE

Un linguaggio  $L \subseteq A^*$  è regolare se e solo se è finito o si ottiene da linguaggi finiti attraverso un numero finito di applicazioni delle operazioni  $\cup, \cdot, *$ .

## DIMOSTRAZIONE

$\Leftarrow$

Immediata poiché i linguaggi finiti sono regolari e dalle proprietà di chiusura già viste

$\Rightarrow$

Sia  $m = (Q, A, \delta, q_1, F)$  un DFA che accetta  $L$ , con  $Q = \{q_1, \dots, q_n\}$

Siamo  $1 \leq i, j \leq n$  e  $k \leq n$ , definiamo

$R_{i,j}^{(k)} = \left\{ w \in A^* \mid \delta^*(q_i, w) = q_j \wedge \delta^*(q_i, w') \in \{q_1, \dots, q_k\} \text{ se } w' \text{ è un} \right.$

prefisso non vuoto e proprio ( $\neq w$ ) di  $w\}$

In particolare per  $k=0$  ottieniamo

$R_{i,i}^{(0)} = \{\epsilon\} \cup \{a \in A \mid \delta(q_i, a) = q_i\}$  (finito) e per  $i \neq j$

$R_{i,j}^{(0)} = \{a \in A \mid \delta(q_i, a) = q_j\}$  (finito)

Per  $K > 0$

$$R_{i,j}^{(k+1)} = R_{i,j}^{(k)} \cup R_{i,k+1}^{(k)} \cdot (R_{k+1,k+1}^{(k)})^* \cdot R_{k+1,j}^{(k)}$$

Dunque tutti gli insiem  $R_{i,j}^{(k)}$  si ottengano da linguaggi finiti  
usando un numero finito di volte le operazioni  $\cup, \cdot, ^*$ .

Ma  $L = L(m) = \bigcup_{j \in F} R_{i,j}^{(m)}$  dunque anche  $L$  si ottiene allo stesso modo

$$V = 5 \text{ m/s}$$

$$5 \text{ m}_1 + \dots = (m_1 + 3) 2,5$$

$$m_2 = 3 \text{ kg}$$

$$2 m_1 = m_1 + 3$$

$$V_2 = 2,5 \text{ m/s}$$

$$2 m - m_1 = 3$$

$$m_1 = ?$$

$$m_1 = 3 \text{ kg}$$

$$\Delta K = ?$$

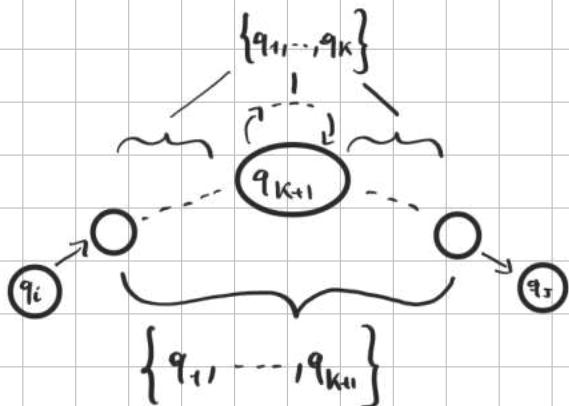
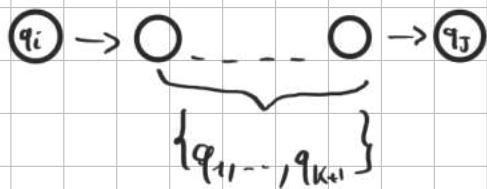
$$K_1 = \frac{1}{2} 3 \cdot 5^2 = 37,5 \text{ J}$$

$$K_2 = \frac{1}{2} (3+3) \cdot 2,5^2 = 18,75 \text{ J}$$

$$\Delta K = 37,5 - 18,75 = 18,75 \text{ J}$$

## SPIEGAZIONE TEOREMA DI KLEENE

$$R_{i,j}^{(k+1)} = R_{i,j}^{(k)} \cup R_{i,k+1}^{(k)} \cdot (R_{k+1,k+1}^{(k)})^* \cdot R_{k+1,j}^{(k)}$$



## ESPRESSIONI REGOLARI

Sia  $A = \{a_1, \dots, a_k\}$  e poniamo  $\hat{A} = \{a_1, \dots, a_k, \emptyset, \epsilon, \cup, \cdot, ^*, (), ()\}$

Un'espressione regolare su  $A$  è una parola di  $\hat{A}^*$  appartenente al sottinsieme definito dalle regolari regole (**SINTASSI**):

- 1)  $a_1, \dots, a_k, \epsilon$  e  $\emptyset$  sono espressioni regolari
- 2) Se  $\alpha, \beta$  sono espressioni regolari lo è anche  $(\alpha \cup \beta)$
- 3) Se  $\alpha, \beta$  sono espressioni regolari lo è anche  $(\alpha \cdot \beta)$
- 4) Se  $\alpha$  è espressione regolare, lo è anche  $\alpha^*$

## SEMANTICA

Ad ogni espressione regolare  $\alpha$  su  $A$  corrisponde un linguaggio

$\langle \alpha \rangle \subseteq A^*$  secondo quanto segue:

1) per  $i=1, \dots, K$   $\langle \alpha_i \rangle = \{a_i\}$

2)  $\langle \emptyset \rangle = \emptyset$  e  $\langle \varepsilon \rangle = \{\varepsilon\}$

3) Se  $\alpha$  e  $\beta$  sono espressioni regolari,  $\langle (\alpha \cup \beta) \rangle = \langle \alpha \rangle \cup \langle \beta \rangle$

4) Se  $\alpha$  e  $\beta$  sono espressioni regolari,  $\langle \alpha \cdot \beta \rangle = \langle \alpha \rangle \cdot \langle \beta \rangle$

5) Se  $\alpha$  è espressione regolare,  $\langle \alpha^* \rangle = \langle \alpha \rangle^*$

È possibile scrivere espressioni regolari semplificate, eliminando parentesi non necessarie e ":" , utilizzando le regole di precedenza delle operazioni

$* > \cdot > \cup$

Per il Teorema di Kleene, un linguaggio  $L \subseteq A^*$  è regolare se e solo se esiste un' espressione regolare  $\alpha$  tale che  $\langle \alpha \rangle = L$

E.

- Sia  $\alpha$  formato dalle parole in  $\{a, b\}$  che contengono  $aa$  e terminano in  $b$

Allora  $L = \langle \underbrace{((a \cup b)^* \cdot (a \cdot a) \cdot (a \cup b)^*)}_b \rangle = \langle (a \cup b)^* aa (a \cup b) b \rangle$

tutte le possibili concatenazioni

Utilizziamo  $\leftrightarrow$  nei linguaggi regolari.



$$L(m) = R_{1,2}^{(2)}$$

$$R_{1,2}^{(2)} = R_{1,2}^{(1)} \cup R_{1,2}^{(1)} \cdot (R_{2,2}^{(1)})^* \cdot R_{2,2}^{(1)}$$

$$R_{1,2}^{(1)} = \langle (a^* \cdot b) \rangle = \langle a^* b \rangle$$

$$R_{2,2}^{(1)} = \langle (\epsilon \cup b) \cup (a(a^*b)) \rangle = \langle \epsilon \cup b \cup aa^*b \rangle$$

$$L(m) = \langle a^*b \cup a^*b (\epsilon \cup b \cup aa^*b)^* (\epsilon \cup b \cup aa^*b) \rangle =$$

$$= \langle a^*b (\epsilon \cup (aa^*b)^*)^* (\epsilon \cup b \cup aa^*b) \rangle =$$

$$= \langle a^*b (b \cup aa^*b)^* \rangle \leftarrow \text{soluzione}$$

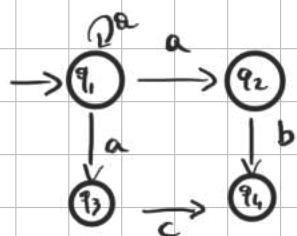
OSS

Qui abbiamo applicato la proprietà **DISTRIBUTIVA**

Un'altra proprietà utile è  $L_1 L_2 \cup \{\epsilon\} = L_1$

Esempio 1 pg 246

$$L(m) = \langle a^*ab \cup a^*ac \rangle$$



## LEMMA DI ITERAZIONE (pumping lemma)

Sia  $m$  un DFA su  $A$  con  $m$  stati e sia  $x \in L(m) \mid |x| \geq m$

Allora  $\exists u, v, w \in A^* \mid x = uvw$

$$v \neq \epsilon$$

$$\forall i \geq 0, uv^i w \in L(m)$$

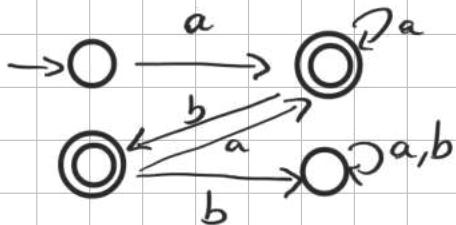
Esprire una condizione **NECESSARIA** affinché un linguaggio sia regolare



## Esercizi

pj 258 n° 5

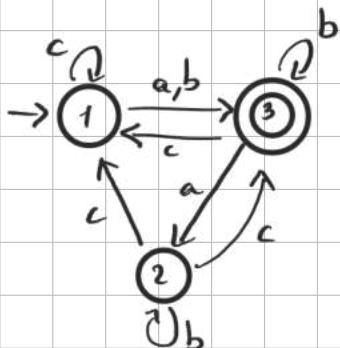
$$L = \{x \in \{a, b\}^* \mid x \neq \epsilon \text{ e } x \text{ non contiene } bb\}$$



$$L = \langle b(a \cup ab)^* \cup (a \cup ab)(a \cup ab)^* \rangle = \langle (b \cup a \cup ab)(a \cup ab)^* \rangle'$$

Esercizio 3c

$$L(m) = R_{1,3}^{(3)}$$



$$R_{1,3}^{(3)} = R_{1,3}^{(2)} \cup R_{1,3}^{(2)} \cdot (R_{3,3}^{(2)})^* \cdot R_{3,3}^{(2)}$$

$$R_{1,3}^{(2)} = R_{1,3}^{(1)} = \langle c^*(a \cup b) \rangle$$

$$R_{3,3}^{(2)} = R_{3,3}^{(1)} \cup R_{3,2}^{(1)} \cdot (R_{2,2}^{(1)})^* \cdot R_{2,3}^{(1)} =$$

$$= \langle \epsilon \cup b \cup c^*(a \cup b) \cup a(\epsilon \cup b)^*(c \cup ac^*(a \cup b)) \rangle =$$

$$= \langle \epsilon \cup b \cup ab^*c \cup (c \cup ab^*a)c^*(a \cup b) \rangle = \langle \alpha \rangle$$

$$L(m) = \langle c^*(a \cup b)(\epsilon \cup a^*a) \rangle = \langle c^*(a \cup b)a^* \rangle = \langle c^*(a \cup b)(b \cup ab^*a)c^*(a \cup b) \rangle^*$$

## PUMPING LEMMA PER LINGUAGGI REGOLARI

Sia  $\mathcal{A}$  un DFA su  $A$  con  $n$  stati e  $x \in L(\mathcal{A})$  |  $|x| \geq n$

Allora  $\exists u, v, w \in A^*$  |

- ①  $x = uvw$
- ②  $v \neq \epsilon$
- ③  $\forall i \geq 0 \quad uv^i w \in L(\mathcal{A})$

### DIMOSTRAZIONE

Usiamo il "principio della piccionaia":

Se abbiamo un numero  $\geq n+1$  di oggetti da distribuire in  $n$  insiem, allora almeno uno di tali insiem contiene almeno 2 oggetti.

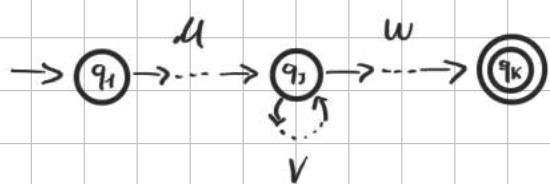
Dato che  $|x| \geq n$ , il numero totale di passaggi per uno stato lungo il cammino etichettato con  $x$  è  $\geq n$  (a partire dallo stato iniziale) e' almeno  $n+1$ ; dunque esiste almeno uno stato per cui passiamo almeno due volte. Formalmente:

Se  $Q = \{q_1, \dots, q_m\}$  è l'insieme degli stati di  $\mathcal{A}$ ,

abbiamo  $S^*(q_1, x) = q_k \in F$ ,

e se  $x = x_1 \dots x_m$  con  $m \geq n$   $\exists h_1, h_2 \mid 1 \leq h_1 < h_2 \leq m$  e

$S^*(q_1, x_1 \dots x_{h_1}) = q_j = S^*(q_1, x_1 \dots x_{h_2})$  (passiamo almeno 2 volte per  $q_j$ )



Sia allora  $u = x_1 \cdots x_{h_1}$ ,  $v = x_{h_1+1} \cdots x_{h_2}$ ,  $w = x_{h_2+1} \cdots x_m$

E' evidente che  $x = uvx$ .

Inoltre  $v \neq \epsilon$  dato che  $h_1 < h_2$

Infine  $\forall i \geq 0$ ,  $s^*(q_1, uv^i w) = q_k \in F$ , per cui  $uv^i w \in L(m)$

NB:  $\forall i \geq 0$ ,  $uv^i w \in L(m) \iff \{u\} \{v\}^* \{w\} \subseteq L(m)$   
 $\langle uv^* w \rangle$

### COROLLARIO

Sia  $M$  un DFA su  $A$  con  $n$  stati. Se  $L(m) \neq \emptyset$ , allora  $L(m)$  contiene parole di lunghezza  $< m$

### DIMOSTRAZIONE

Se per assurdo  $m$  accettasse solo parole di lunghezza  $\geq m$ , prendendo  $x \in L(m)$  di lunghezza minima, dal pumping lemma avremo  $x = uvw$  con  $v \neq \epsilon$  e  $uvw \in L(m)$  ma questo e' un ASSURDO perch'e  $|uv| < |x|$ .

## OSS

Possiamo usare questo risultato anche per verificare, dati due DFA  $m_1, m_2$ , se  $L(m_1) \subseteq L(m_2)$ ; basta costruire un DFA che accetti  $L(m_1) \setminus L(m_2) = L(m_1) \cap \overline{L(m_2)}$ . Se questo non accetta parole, allora  $L(m_1) \subseteq L(m_2)$ .

## PROPOSIZIONE

Sia  $m$  un DFA con  $m$  stati, allora

$L(m)$  è infinito  $\Leftrightarrow L(m)$  contiene parole  $x$  di lunghezza  $m \leq |x| < 2m$

## DIMOSTRAZIONE

$\Leftarrow$

Se  $x \in L(m)$  e  $m \leq |x| < 2m$ , per il pumping lemma  $x = uvw$

$v \neq \epsilon$  e  $uv^iw \in L(m)$  per  $i \geq 0$ , quindi  $L(m)$  è infinito

$\Rightarrow$

Sia  $x \in L(m)$  di lunghezza  $\geq 2m$  ma più corta possibile

Scriviamo  $x = yz$  con  $|y| = m$  e  $|z| \geq m$ .

Applicando il principio della piccionaia come nel pumping lemma,

otteniamo che  $y=uvw$  con  $v \neq \epsilon$  e  $\forall i \geq 0 \quad uv^iw \in L(m)$

In particolare  $uwz \in L(m)$ , e  $|uwz| \geq |z| \geq n$  e inoltre

$$|uwz| < 2n \text{ poiché } |uwz| < |uvwz| = |x|$$

E.

Il linguaggio  $\{a^n b^n \mid n \geq 0\}$  **NON** è regolare

E' intuitivo perché un eventuale DFA che lo accetti dovrebbe avere infiniti stati.

Per dimostrarlo formalmente possiamo usare il pumping lemma.

Se per assurdo  $L$  fosse regolare sarebbe accettato da un DFA con  $p \geq 1$  stati.

La parola  $x=a^p b^p$  dovrebbe poter essere scritta come  $x=uvw$  con  $V \neq \epsilon$  e  $uv^iw \in L$  per  $i \geq 0$ .

Se  $V=a^j$  per  $j > 0$ ,  $uv^2w=a^{p+j}b^p \notin L(m)$

Se  $V=b^j$  per  $j > 0$ ,  $uv^2w=a^p b^{p+j} \notin L(m)$

Se  $V=a^j b^k$  per  $j, k > 0$ ,  $uv^2w=a^{p-j}(a^j b^k)^2 b^{p-k} = a^p b^k a^j b^p \notin L(m)$

E.

- $L_1 = \{a^n b^n \mid n \geq 0\}$  non è regolare

Infatti sia per assurdo un DFA con  $p > 0$  stati tali che  $L_1 = L(M_1)$

(considerando  $x_1 = a^p b^p \in L_1$ , si ha  $|x_1| \geq p$  e una decomposizione

$x_1 = uvw$  con  $V \neq \emptyset$ , i casi possibili sono:

1) V contiene solo a ( $v = a^j$  per qualche  $j > 0$ )

2) V contiene solo b ( $v = b^j$  per qualche  $j > 0$ )

3)  $v = a^i b^k$  per  $i, k > 0$

In tutti e 3 i casi  $uvw \notin L_1$ , quindi non è regolare

- $L_2 = \{a^m b^m \mid m \geq 0\}$  non è regolare

Supponiamo p.a.  $M_2$  un DFA con  $p$  stati tali che  $L_2 = L(M_2)$

Seguiamo la stessa parola di prima  $x_2 = a^p b^p$ , vogliamo gli stessi casi di prima per la decomposizione  $x_2 = uvw$

Nel primo caso  $v = a^j$ , consideriamo  $uv^0 w = vw = a^{p-j} b^p \notin L_2$ ,

gli altri due casi sono analoghi a quelli precedenti

•  $L_3 = \{x \in \{a,b\}^* \mid x = \tilde{x}\}$  linguaggio dei PALINDROMI su  $\{a,b\}$ , non è regolare

Non è possibile dimostrarlo attraverso il pumping lemma come visto

finora: infatti qualsiasi palindromo non vuoto  $x$  si può decomporre come

$x = uvw$  dove  $|v| \leq 2$  e  $w = \tilde{v}$  e allora

$\forall i \geq 0, uv^i w \in L_3$

### PUMPING LEMMA (ancora)

Riprendendo la dimostrazione del pumping lemma,

Sia  $m$  un DFA con  $n$  stati e  $x \in L(m)$  con  $|x| \geq m$ , con  $x = x_1 \dots x_m$  e  $m \geq n$

Poiché  $\{\delta^*(q_0, x_1 \dots x_k) \mid 0 \leq k \leq m\}$  contiene al massimo  $n$  stati distinti, esistono  $j_1 < j_2$  tali che  $\delta^*(q_0, x_1 \dots x_{j_1}) = \delta^*(q_0, x_1 \dots x_{j_2})$ .

Scelgiamo di MINIMALI, cioè: gli stati

$\{\delta^*(q_0, x_1 \dots x_k) \mid 0 \leq k < j_2\}$  sono tutti diversi, per cui  $j_2 \leq m$

e quindi se poniamo  $u = x_1 \dots x_{j_1}$ ,  $v = x_{j_1+1} \dots x_{j_2}$ ,  $w = x_{j_2+1} \dots x_m$

avremo:

1)  $v \neq \epsilon$

2)  $\forall i \geq 0, uv^i w \in L(m)$

3)  $|uv| = m$

E.s.

Per dimostrare che  $L_3$  non è regolare, sia p.a.  $M_3$  un DFA con  $p$  stati tale che  $L_3 = L(M_3)$  e consideriamo  $x_3 = (ab)^p(ba)^p \in L_3$

$$\text{Si ha } |x_3| = 4p > p$$

Consideriamo le decomposizioni  $x_3 = uvw$  con  $V \neq \emptyset$  e  $|uvw| \leq p$

Qualunque sia la scelta di  $V$ ,  $uv$  non può essere palindromo poiché contiene  $bb$  nelle sue prime  $2p$  lettere ma non nelle ultime  $2p$  lettere

Dunque il pumping lemma "esteso" non è verificato, **Arrendo!**

Conclusione:  $M_3$  non può esistere, quindi  $L_3$  non è regolare

### GRAMMATICHE CONTEXT-FREE

Sono nate per l'analisi di linguaggi naturali, sono state poi adattate ai linguaggi formali (sostituzioni di  $A^*$  qualsiasi) e ad esempio ai linguaggi di programmazione

Una **CFG** è una 4-pla  $G = (V, \Sigma, R, S)$  dove

- $V$  è un alfabeto finito (delle **VARIABILI**)
- $\Sigma$  è un alfabeto finito (dei **TERMINALI**)
- $R \subseteq V \times (V \cup \Sigma)^*$  i cui elementi sono detti **REGOLE**

•  $S \in V$  e' detta variabile iniziale o **ASSIOMA**

Una regola  $(X, u)$  si indica normalmente con  $X \rightarrow u$

Date due stringhe  $u, w \in (V \cup \Sigma)^*$ , diciamo che  $u$  **PRODUCE**  $w$  (per

**DERIVAZIONE ELEMENTARE** secondo  $G$ ) se  $R$  contiene una regola  $X \rightarrow v$

talche  $u = u_1 X u_2$  e  $w = u_1 v u_2$  per qualche  $u_1, u_2 \in (V \cup \Sigma)^*$

Scriviamo  $u \xrightarrow[G]{} w$

Scriviamo invece  $u \xrightarrow[G]^* w'$  se  $\exists n \geq 0, u_1, u_2, \dots, u_n$  tali che

$u \xrightarrow[G]{} u_1 \xrightarrow[G]{} \dots \xrightarrow[G]{} u_n = w'$ , diciamo che  $w'$  si ottiene per derivazione secondo  $G$ .

Allora il **LINGUAGGIO GENERATO** sara'  $L(G) = \left\{ x \in \Sigma^* \mid S \xrightarrow[G]^* x \right\}$

Ej.

$G = (V, \Sigma, R, S)$  con  $V = \{S\}, \Sigma = \{a, b\}, R = \underline{\{S \rightarrow aSb, S \rightarrow \epsilon\}}$

$S \rightarrow aSb \mid \epsilon$

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

Infatti una **DERIVAZIONE** per  $a^n b^n$  e':

$S \xrightarrow[]{} aSb \xrightarrow[]{} aaSbb \xrightarrow[]{} \dots \xrightarrow[]{} a^n S b^n \xrightarrow[]{} a^n b^n$

lio dimostra che tra i linguaggi context-free (cioe' generati da una CFG) ce ne sono di **NON** regolari

E.s.

$$S \rightarrow aSb \mid \epsilon$$

Derivazione elementare

$$u \Rightarrow v \quad x \quad \exists x \Rightarrow_{w \in R} \exists u_1, u_2 \in (V \cup \Sigma)^* \mid u = u_1 X u_2, v = u_1 w u_2$$

### DERIVAZIONE

Sequenza di zero o più derivazioni elementari ( $x \xrightarrow[G]{*} y$ )

$$L(G) = \left\{ x \in \Sigma^* \mid S \xrightarrow[G]{*} x \right\}$$

$$L_2 = \left\{ a^m b^m \mid m \geq m > 0 \right\}$$

$$a^n b^m = a^{n-m} a^m b^m$$

$$S \rightarrow AX$$

$$A \rightarrow_a A \mid \epsilon$$

$$X \rightarrow_a Xb \mid ab$$

$$a^4 b^3 \in L_2$$

$$S \Rightarrow AX \Rightarrow_a AX \Rightarrow_a X \Rightarrow_a a a a X b b \Rightarrow a a a b b b$$

### OSS

Una regola come  $S \rightarrow AX$  puo' essere interpretata come equazione su

linguaggio: cioè  $L_2 = L_A \cup L_X$ , con  $L_A = \{a^k \mid k \geq 0\}$  e  $L_X = \{a^m b^m \mid m > 0\}$

L.

$$L_X = \{a\} \cup \{b\} \cup \{ab\}$$

Consideriamo la grammatica  $G_1$  con regole:

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

$$T \rightarrow T \cdot F \mid F$$

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

dove  $G_1 = (V_1, \Sigma_1, R_1, E)$  con  $V_1 = \{E, T, F\}$ ,  $\Sigma_1 = \{a, +, \cdot, (), \}\}$

Ad esempio

$$a+a \cdot a \in L_1(G_1)$$

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow T + T \cdot F \Rightarrow T + F \cdot F \Rightarrow F + F \cdot F \Rightarrow a + F \cdot F \Rightarrow a + a \cdot F \Rightarrow a + a \cdot a$$

Non c'è l'unica derivazione, avremmo potuto scrivere

$$T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T \cdot F \Rightarrow a + F \cdot F \Rightarrow \dots$$

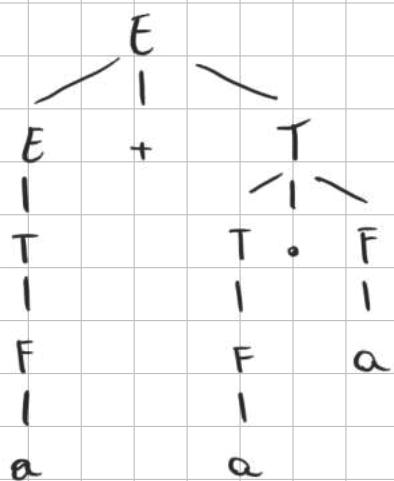
Cioè che cambia è solo l'ordine delle regole, quindi non vi c'è una vera e propria ambiguità

## ALBERO DI DERIVAZIONE

Data una derivazione in G, costruiamo un albero che ha come **RADICE** la variabile iniziale, e tale che a ogni passo della derivazione corrispondono tanti figli della variabile da sostituire quante sono le lettere della stringa che la sostituisce.

E.s.

Vediamo l'esempio di prima



## DERIVAZIONI EQUIVALENTI

Due derivazioni sono equivalenti se corrispondono allo stesso albero, o equivalentemente alla stessa derivazione **ESTREMA SINISTRA** (in cui ad ogni passo la variabile da sostituire è la prima, per evitare una finita ambiguità)

## GRAMMATICA AMBIGUA

Una grammatica  $G$  si dice **AMBIGUA** se esiste  $x \in L(G)$  per la quale ci sono più derivazioni non equivalenti.

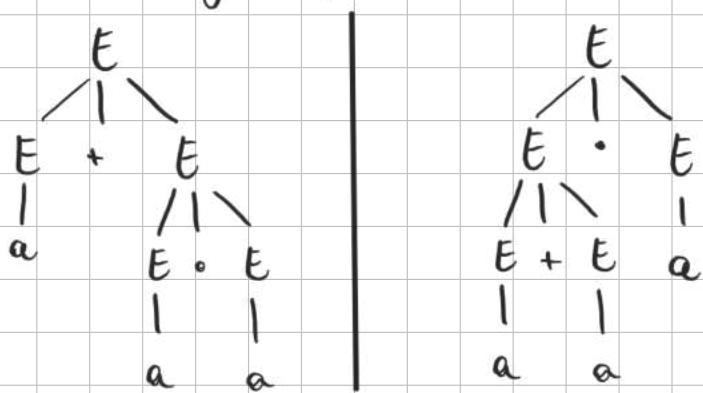
Ad esempio  $G_1$  non è ambigua

Ese.

$$G_2 = E \rightarrow E+E \mid E \cdot E \mid (E) \mid a$$

Si ha  $L(G_2) = L(G_1)$

$G_2$  è ambigua perché:



$$E \Rightarrow E+E \Rightarrow a+a \Rightarrow E \Rightarrow E \cdot E \Rightarrow E+E \cdot E \Rightarrow a+E \cdot E = a+a \cdot E \Rightarrow a+a \cdot a$$

$$\Rightarrow a+E \cdot E \Rightarrow a+a \cdot E \Rightarrow$$

$$\Rightarrow a+a \cdot a$$

OSS

C'è una vaga correlazione tra l'ambiguità e il non determinismo, che sono comunque concetti distinti e indipendenti tra loro.

Esistono linguaggi CF che possono essere generati **SOLO** da grammatiche ambigue e sono detti **linguaggi INERENTEMENTE AMBIGUI**

### PROPRIETÀ DI CHIUSURA ①

Se  $L_1$  e  $L_2$  sono linguaggi CF, allora  $L_1 \cup L_2$  è CF

#### DIMOSTRAZIONE

Siamo  $G_1 = (V_1, \Sigma, R_1, S_1)$  e  $G_2 = (V_2, \Sigma, R_2, S_2)$  grammatiche tali che

$$L_1 = L(G_1), \quad L_2 = L(G_2), \quad V_1 \cap V_2 = \emptyset$$

Allora la grammatica  $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, R, S)$

con  $R = R_1 \cup R_2 \cup \{S \rightarrow S_1 \mid S_2\}$  genera  $L_1 \cup L_2$

### PROPRIETÀ DI CHIUSURA ②

Se  $L_1$  e  $L_2$  sono linguaggi CF, allora  $L_1 L_2$  è CF

#### DIMOSTRAZIONE

Consideriamo  $G_1$  e  $G_2$  come prima

Allora  $G' = (V, \Sigma, R', S')$  con  $R' = R_1 \cup R_2 \cup \{S' \rightarrow S_1 S_2\}$

genera  $L_1 L_2$

OSS

Ricordiamo che se  $L \subseteq \Sigma^*$ ,  $L^* = L^* L \cup \{\epsilon\}$

### PROPRIETÀ DI CHIUSURA (3)

Se  $L_1$  è un linguaggio CF, lo è anche  $L_1^*$

### DIMOSTRAZIONE

Sia  $G_1$  come prima, consideriamo  $G_0 = (V, U \{S_0\}, \Sigma, R_0, S_0)$ , con  $R_0 = R_1 \cup \{S_0 \rightarrow S_0 S_1 \mid \in\}$ , essa genera  $L_1^*$

### PROPOSIZIONE

Se  $L \subseteq \Sigma^*$  è finito, allora è CF

### DIMOSTRAZIONE

Sia  $L = \{w_1, \dots, w_m\}$ .

La grammatica le cui regole sono  $S \rightarrow w_1 \mid \dots \mid w_m$  genera  $L$

### COROLLARIO

Se  $L$  è regolare, è CF

### DIMOSTRAZIONE

Dal teorema di Kleene,  $L$  si ottiene da linguaggi finiti (e quindi CF) attraverso un numero finito di operazioni di  $\cup$ ,  $\cdot$  e  $*$

La teoria segue dalle proposizioni di chiusura

E.

Sia  $L = \{a^i b^j c^k \mid i, j, k \geq 0 \wedge (i=j \vee i=k)\}$

Dimostriamo che  $L$  e' CF.

Si ha  $L = L_1 \cup L_2$  con  $L_1 = \{a^m b^m c^k \mid m, k \geq 0\}$  e  $L_2 = \{a^m b^j c^m \mid j, m \geq 0\}$

$S_1 \rightarrow XC$

$X \rightarrow aXb \mid \epsilon$

$C \rightarrow cC \mid \epsilon$

$S_2 \rightarrow aS_2c \mid B$

$B \rightarrow bB \mid \epsilon$

Tale grammatica genera  $L$  ed e' ambigua

$S \Rightarrow S_1 \Rightarrow XC \Rightarrow aXbC \Rightarrow abC \Rightarrow abcC \Rightarrow abc$

Sono derivazioni estreme a sinistra per  $abc \in L$

E.

Regole:

$\langle \text{Prog} \rangle \rightarrow \epsilon \mid \langle \text{Lista Istruzioni} \rangle \langle \text{NonPignaY} \rangle$

$\langle \text{Lista Istruzioni} \rangle \rightarrow \langle \text{Istruz} \rangle \mid \langle \text{Istruz} \rangle \langle \text{Lista Istruz} \rangle$

$\langle \text{Istruz} \rangle \rightarrow \langle \text{NonPignaY} \rangle \mid Y$

$\langle \text{NonPignaY} \rangle \rightarrow \langle \text{Var} \rangle \text{++} \mid \langle \text{Var} \rangle \text{--} \mid \langle \text{NonY} \rangle \mid \langle \text{Salto} \rangle \mid [\text{label}] \langle \text{Var} \rangle \text{++} \mid$   
 $[\text{label}] \langle \text{Var} \rangle \text{--} \mid [\text{label}] \langle \text{Var} \rangle \mid [\text{label}] \langle \text{Salto} \rangle$

$\langle \text{Salto} \rangle \rightarrow \text{IF } \langle \text{Var} \rangle \neq 0 \text{ GOTO } \langle \text{label} \rangle \mid [\text{label}] \text{ IF } \langle \text{Var} \rangle \neq 0 \text{ GOTO } \langle \text{label} \rangle$

$\langle \text{Var} \rangle \rightarrow \langle \text{NonY} \rangle \mid Y$

$\langle \text{NonY} \rangle \rightarrow X : \langle \text{Num} \rangle \mid Z : \langle \text{Num} \rangle$

$\langle \text{label} \rangle \rightarrow A : \langle \text{Num} \rangle \mid B : \langle \text{Num} \rangle \mid \dots \mid E : \langle \text{Num} \rangle$

$\langle \text{Num} \rangle \rightarrow \langle \text{NonO} \rangle \mid \langle \text{NonO} \rangle \langle \text{Lista Cifre} \rangle$

$\langle \text{Lista Cifre} \rangle \rightarrow \langle \text{Cifra} \rangle \mid \langle \text{Cifra} \rangle \langle \text{Lista Cifre} \rangle$

$\langle \text{Cifra} \rangle \rightarrow \langle \text{NonO} \rangle \mid 0$

$\langle \text{NonO} \rangle \rightarrow 1 \mid \dots \mid 9$

Soluzione es. 2.4

b)  $S \rightarrow 0|1|0x0|1x1$

$X \rightarrow 0X|1X|\epsilon$

c)  $S \rightarrow 0|1|0p|1p$

$P \rightarrow \epsilon|00p|01p|10p|11p$

d)  $S \rightarrow 0|0s0|0s1|1s0|1s1$

Aggiungendo "11" si ha un'altra soluzione per c

e)  $S \rightarrow \epsilon|0|1|0s0|1s1$

### GRAMMATICHE REGOLARI

Sia  $m = (Q, \Sigma, S, q_1, F)$  un DFA e costruiamo un CFG  $G$  tale che

$$L(G) = L(m).$$

Sia  $V$  un insieme di variabili della stessa cardinalità di  $Q$ ;

chiamiamo  $X_i \in V$  la variabile corrispondente a  $q_i \in Q$

Definiamo  $G = (V, \Sigma, R, X_i)$ , dove  $R$  contiene le seguenti regole:

Se  $\delta(q_i, a) = q_j$ , allora  $X_i \rightarrow aX_j \in R$ ;

Se  $q_i \in F$  allora  $X_i \rightarrow \epsilon$

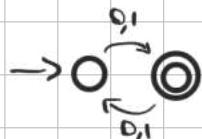
(In alternativa: se  $q_i \in F \setminus \{q_n\}$  e  $\delta(q_n, a) = q_i$ , anche  $X_n \rightarrow a$ )

Allora  $L(G) = L(m)$ .

In generale una grammatica si dice **REGOLARE** se tutte le regole hanno la forma  $X \rightarrow aY$  oppure  $X \rightarrow a$  oppure  $X \rightarrow \epsilon$

E.

l'automa dell'esercizio 2.4c:



Corrisponde alla grammatica regolare:

$$S \rightarrow 0x11x$$

$$S \rightarrow 0x11x1011$$

$$X \rightarrow \emptyset S | 1 S | \epsilon$$

oppure

$$X \rightarrow 0 S | 1 S$$

AUTOMI PUSH DOWN o a pila

Aggiungono alle normali operazioni previste, una forma di input / output su una memoria (pila) non limitata, secondo il paradigma LIFO.

Ad ogni passo, le operazioni possibili sono:

- Lettura della prossima lettera della parola di input
- Estrazione del primo simbolo della pila (**POP**)

- Scriviamo di un simbolo in cima alla pila (**PUSH**)

Dopo di che si passa agli stati successivi.

Formalmente un **PDA** è una 6-pla  $(Q, \Sigma, \Gamma, \delta, q_0, F)$  con

- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow P(Q \times (\Gamma \cup \{\epsilon\}))$  funzione di transizione

- $\Sigma$  alfabeto della stringa di input,

- $\Gamma$  alfabeto dei simboli sulla pila,

- $q_0$  stato iniziale

- $F$  stati terminali



## PAROLE ACCETTATE

Una parola  $w \in \Sigma^*$  e' accettata da un pilota se

$w = w_1 \dots w_m$ ,  $m \geq 0$  e  $w_i \in \Sigma \cup \{\epsilon\}$  per  $i = 1, \dots, m$

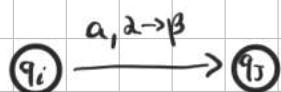
$\exists q_1, \dots, q_{m+1} \in Q$ ,  $s_0 = \epsilon$ ,  $s_1, \dots, s_m \in \Gamma^*$  tali che per  $i = 1, \dots, m$

$(q_i, w_i, s_i) \in \delta(q_i, w_i, s_i)$ , e  $s_i = \alpha_i t_i$ ,  $s_{i+1} = \beta_i t_i$  per

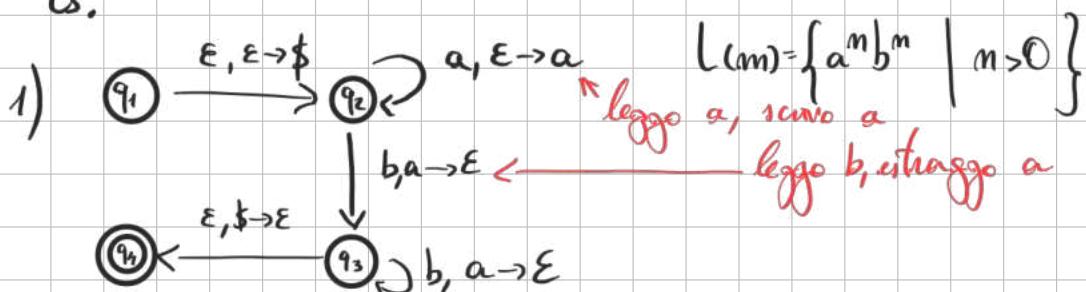
$t_i \in \Gamma^*$ ,  $\alpha_i, \beta_i \in \Gamma \cup \{\epsilon\}$ ,  $q_{m+1} \in F$

Le parole  $s_i \in \Gamma^*$  rappresentano tutto ciò che e' scritto nella pila a un dato istante.

$(q_j, p) \in \delta(q_i, a, s)$  si rappresenta nel diagramma con



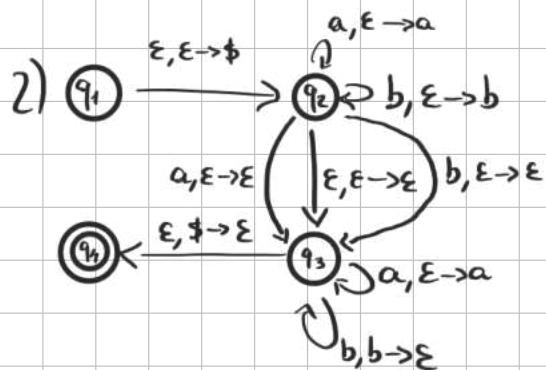
E.



Il simbolo  $\$$  viene inserito all'inizio. Quando viene rimosso significa che la pila e' vuota (funziona da flag), per questo  $\Sigma \neq \Gamma$

OSS

ha pila accetta linguaggi non regolari



$$L(m_2) = \{w \in \{a, b\}^* \mid w = \bar{w}\}$$

Pg 163 mo 2.3, 2.10

Scrivere le regole di una CFG che genera

$$L = \{a^i b^j c^k \mid i=j \vee j=k, i, j, k \geq 0\}$$

$$S \rightarrow Xc \mid AY$$

$$X \rightarrow aXb \mid \epsilon$$

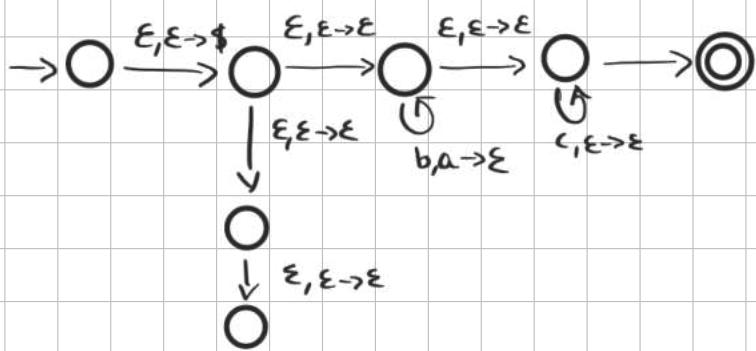
$$Y \rightarrow bXc \mid \epsilon$$

$$Y \rightarrow bXc \mid \epsilon$$

$$A \rightarrow aA \mid \epsilon$$

$$C \rightarrow cC \mid \epsilon$$

Disegnare il diagramma di un PDA che accetti L



### TEOREMA

$L \subseteq \Sigma^*$  context free  $\Leftrightarrow L$  accettato da un PDA

### NOTAZIONE $\delta^*$

Siamo  $a \in \Sigma \cup \{\epsilon\}$ ,  $\beta \in \Gamma \cup \{\epsilon\}$  e  $w = a_1 \dots a_m \in \Gamma^*$  ( $a_i \in \Gamma$  per  $i=1, \dots, m$ ),

se  $\exists q_1 = q', \dots, q_m \in Q : (q_m, a_m) \in \delta(q, a, \beta)$  e

$$\delta(q_m, \epsilon, \epsilon) = \{(q_{m-1}, a_{m-1}), \dots, \delta(q_1, \epsilon, \epsilon) = \{(q'_1, a_1)\} \text{ per } i=1, \dots, m$$

Avremo  $(q, w) \in \delta^*(q, a, \beta)$ .

Nel diagramma corrisponde a:



Possiamo passare da  $q$  a  $q'$  tramite una serie di stati intermedi

leggendo  $a$  come input, sostituendolo  $\beta$  con la parola  $w$  (a ritroso, così da leggerlo correttamente)

$$\Rightarrow q \xrightarrow{\alpha, \beta \rightarrow \omega} q'$$

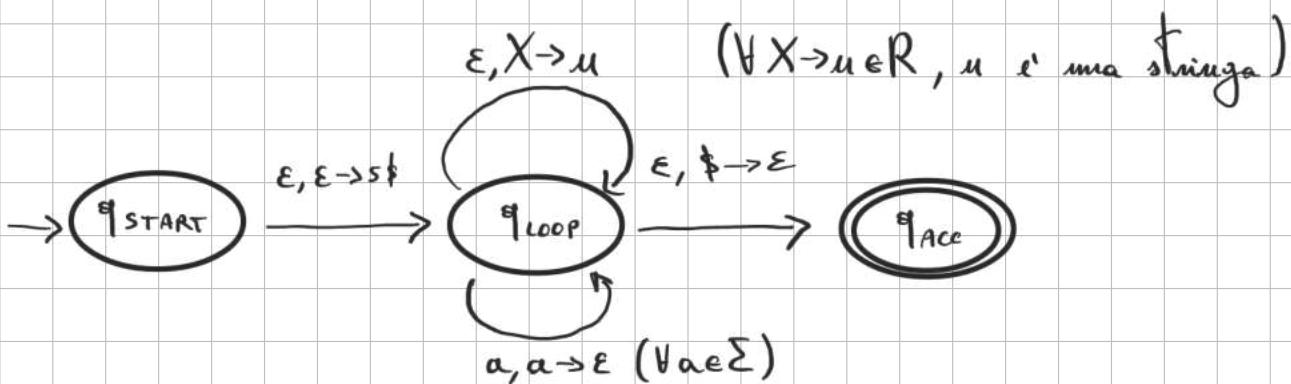
## DIMOSTRAZIONE (TEOREMA)

$\Rightarrow$

Se  $L$  è CF e' accettato da un PDA

Sia  $G = (V, \Sigma, R, S)$  una CFG tale che  $L = L(G)$

disegniamo il diagramma di un PDA, semplificato secondo la notazione  $\delta^*$



l'alfabeto della pila comprende  $\Sigma, S, V$  e '\$'

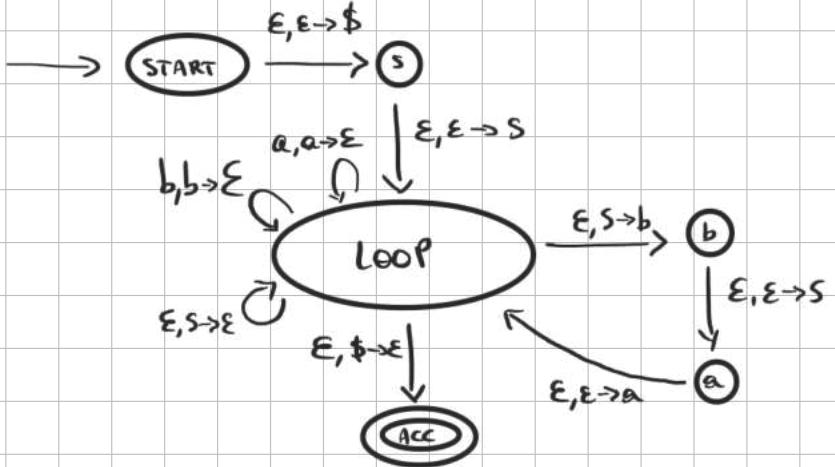
Tale automa accetta esattamente  $L$ .

E.

Consideriamo la grammatica  $G$ , con regole

$$S \rightarrow aSb \mid \epsilon$$

$$L = \{a^n b^n\}$$



Se leggessimo 'aabb' avremo nella pila

$S\$ \rightarrow aSb\$ \rightarrow Sbb\$ \rightarrow aSbb\$ \rightarrow Sbb\$ \rightarrow bb\$ \rightarrow \$ \rightarrow \epsilon$

### OSS

Poiché un NFA è un PDA in cui la pila non viene utilizzata e in ogni transizione viene letta una lettera dell'input, abbiamo una terza dimostrazione del fatto che tutti i linguaggi regolari sono context free.

### PUMPING LEMMA PER CFL

Se  $L \subseteq \Sigma^*$  è CF,  $\exists p \geq 0 \mid \forall w \in L$ ,

$|w| \geq p \Rightarrow \exists u, v, x, y, z \in \Sigma^* \mid w = uvxyz \text{ e}$

①  $\forall i \geq 0, uv^ix^zy^iz \in L$

$$\textcircled{2} |vy| > 0$$

$$\textcircled{3} |vxy| \leq p$$

E.

$$L = \{a^m b^n c^m \mid m \geq 0\} \text{ non e' CF}$$

Infatti, se per oscurato la sia, e sia p la lunghezza di pumping

Consideriamo  $a^p b^p c^p \in L$  e sia  $a^p b^p c^p = uvxyz$  con  $|vy| > 0$

Ci sono 2 casi:

- $v$  e  $y$  contengono lettere di un solo tipo

Ad esempio se  $v \in \{a\}^*$  e  $y \in \{c\}^*$ , chiaramente  $uv^2x^2y^2z$  non ha più lo stesso numero di  $a, b, c$  e dunque non appartiene a  $L$ .

- $v$  o  $y$  (o entrambe) contengono lettere di più tipi.

Ad esempio  $v = a^j b^k$ , allora  $uv^2x^2y^2z \notin \langle a^* b^* c^* \rangle$  e dunque non appartiene neanche a  $L$ .

Cio' dimostra che il pumping lemma non sarebbe valido; l'assurdo segue dall'aver supposto  $L$  context-free, quindi non lo e'.