

TEOREMA 3.10 (Rabin-Scott, 1959). *Sia  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un NFA. Allora esiste un DFA  $M'$  tale che  $L(M) = L(M')$ .*

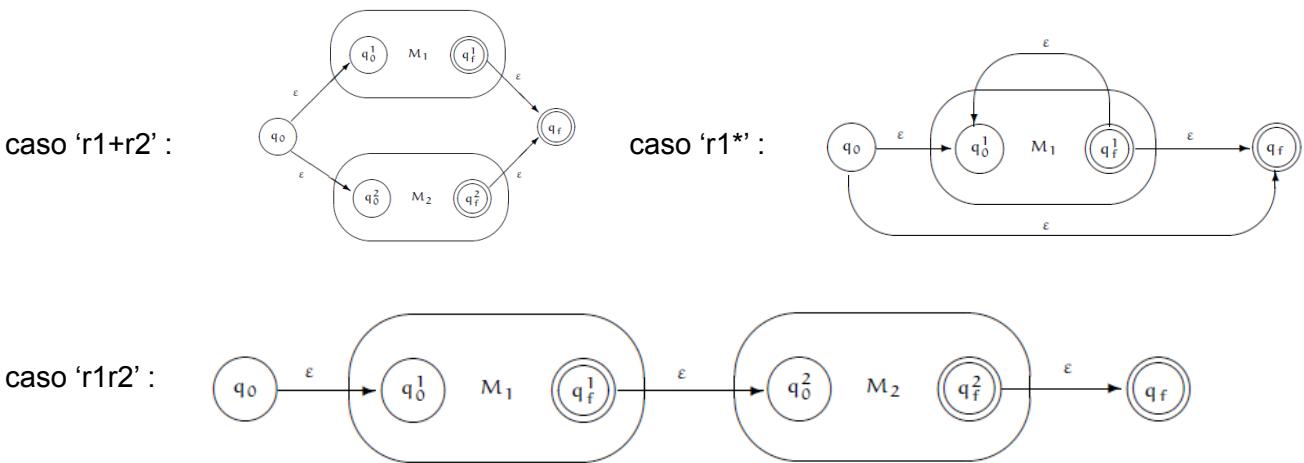
dim: penso deterministicamente all'NFA, ovvero costruisco uno stato per ogni sottoinsieme di stati possibili di  $Q$  (saranno quindi  $\#$  simboli  $\wedge |Q| \rightarrow$  numero di stati del nuovo dfa che è esponenziale rispetto a quelli dell'NFA) e costruisco deterministicamente un DFA. Verifico poi per induzione che la funzione di transizione dei due automati coincida sia nel caso base che nel passo induttivo.

TEOREMA 3.13. *Sia  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un  $\epsilon$ -NFA. Allora esiste un NFA  $M'$  tale che  $L(M) = L(M')$ .*

dim: dato  $\epsilon$ -NFA costruisco un NFA in cui collego lo stato  $qi$  a tutti gli stati che i suoi adiacenti raggiungevano con  $\epsilon$ . Inoltre diventano stati finali anche quelli che con  $\epsilon$  arrivavano in uno stato finale in  $M$ .

TEOREMA 4.4 (McNaughton & Yamada, 1960). *Sia  $r$  una espressione regolare. Allora esiste un  $\epsilon$ -NFA  $M$  tale che  $L(M) = L(r)$ .*

dim: costruisco un  $\epsilon$ -NFA con uno stato finale e per induzione dimostro sulla complessità strutturale di  $r$ . Nel caso di  $\epsilon$  costruisco uno stato solo che è finale, nel caso di linguaggio vuoto costruisco due stati che non sono collegati e nel caso di linguaggio = {a} costruisco uno stato iniziale che se riceve a va in uno stato finale. Da  $r$  posso ottenere un  $\epsilon$ -NFA  $\rightarrow$  NFA  $\rightarrow$  DFA.



TEOREMA 4.6. *Sia  $M$  un DFA. Allora esiste una espressione regolare  $r$  tale che  $L(M) = L(r)$ .*

dim: costruisco il sistema di equazioni per ogni stato e poi risolvendo il sistema ottengo una regex  $r$ .

## PUMPING LEMMA (regolari)

LEMMA 5.1 (Bar-Hillel, Perles, Shamir, 1961). *Sia  $L$  un linguaggio regolare. Allora esiste una costante  $n \in \mathbb{N}$  tale che per ogni  $z \in L$  tale che  $|z| \geq n$  esistono tre stringhe  $u, v, w$  tali che:*

- (1)  $z = uvw$ ,
- (2)  $|uv| \leq n$ ,
- (3)  $|v| > 0$ , e
- (4) per ogni  $i \geq 0$  vale che  $uv^i w \in L$ .

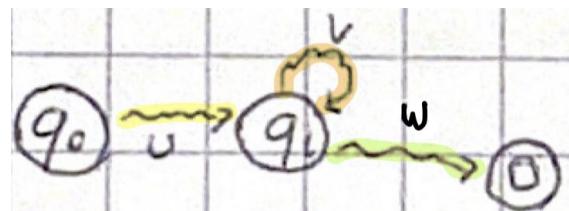
$$\exists n \in \mathbb{N} \forall z \left( \begin{array}{l} (z \in L \wedge |z| \geq n) \rightarrow \exists u, v, w \\ \left( \begin{array}{l} z = uvw \wedge |uv| \leq n \wedge |v| > 0 \wedge \\ \forall i (i \in \mathbb{N} \rightarrow uv^i w \in L) \end{array} \right) \end{array} \right)$$

dim: costruisco un DFA  $M$  con  $n$  stati (da 0 a  $n-1$ ) tale che  $L = L(M)$  e so che:

se  $|z| < n$  allora verificata l'implicazione;

se  $|z| \geq n$  per forza (piccionaia) c'è un ciclo all'interno del dfa perché partendo da  $q_0$  fino a  $q_{n-1}$  percorro  $(n-1)$  stati e la mia stringa è lunga almeno  $n$ .

L'esponente  $i$  che assegno a  $v$  (con  $|v| > 0$ ) indica quante volte percorro il ciclo dentro il DFA.



TEOREMA 5.12 (Vuoto-infinito). *L'insieme delle stringhe accettate da un DFA*

*M con  $n$  stati è:*

- (1) non vuoto se e solo se accetta una stringa di lunghezza inferiore a  $n$ ;
- (2) infinito se e solo se l'automa accetta una stringa di lunghezza  $l$ ,  $n \leq l < 2n$ .

dim:

1. se non accettasse nulla di lunghezza inferiore a  $n$  allora col P.L. io posso spompare qualsiasi stringa e ricadere nell'intervallo  $< n$ .
2. se prendo una stringa  $z$  con  $n \leq |z| < 2n$ , se fosse  $> 2n$  col P.L. posso rientrare nell'intervallo descritto sopra, se fosse  $< n$  non ci potrebbe essere un ciclo e quindi non posso applicare il P.L. Verificate entrambe allora ci sono infinite stringhe generabili col P.L.

TEOREMA 5.17 (Myhill-Nerode, 1957–58). *I seguenti enunciati sono equivalenti:*

- (1)  $L \subseteq \Sigma^*$  è accettato da un qualche DFA;
- (2)  $L$  è l'unione di classi di equivalenza di  $\Sigma^*$  indotte da una relazione invariante a destra e di indice finito;
- (3)  $R_L$  è di indice finito.

dim:

1 → 2.  $L$  è regolare perché accettato da DFA.  $R_M$  ( $x R_M y$  se  $\delta(q_0, x) = \delta(q_0, y)$ ) è la relazione che soddisfa (2) e  $Lq_i$  sono le singole classi di equivalenza (una classe di eq. contiene tutte le stringhe che finiscono nello stato  $q_i$  della partizione indotta da  $R_M$ ). Diciamo che  $L$  è l'unione di queste classi di equivalenza.

2 → 3. Dimostriamo che  $R_M$  raffina  $R_L$ , ovvero che ogni relaz. di eq.  $R$  che soddisfa (2) è un raffinamento di  $R_L$ . Va dimostrato che  $\forall x \in \Sigma^* [x]_R \subseteq [x]_{R_L}$  poiché  $R$  è invariante a dx allora se  $x R y \Rightarrow \forall z \in \Sigma^* xz R yz$  poiché  $L$  è unione di classi di equivalenza (non per forza tutte) di  $R$ , si ha che ogni volta che  $x R y \Rightarrow x \in L$  sse  $y \in L$ , pertanto  $\forall z \in \Sigma^* xz \in L$  sse  $yz \in L$ . Allora otteniamo proprio la definizione di  $R_L$  ( $x R_L y$  sse  $(\forall z \in \Sigma^*) (xz \in L \text{ sse } yz \in L)$ ). L'indice di  $R_L$  è minore di quello di  $R$ , che per ipotesi è finito.

3 →

$M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$

1.

- $Q'$  l'insieme (finito per ipotesi) di classi di equivalenza di  $R_L$ ,
- $\Sigma'$  lo stesso di  $L$ ,
- $\delta'([x], a) = [xa]$  (la definizione ha senso indipendentemente dalla scelta di  $x$  in quanto  $R_L$  è invariante a destra),
- $q'_0 = [\varepsilon]$ ,
- $F' = \{[x] : x \in L\}$ .

Costruire da  $R_L$  un DFA.  $M'$  è il DFA che riconosce  $L$ ,

dove:

(in  $M'$  al posto che gli stati ragiono con le classi di equivalenza come stati)  
e si mostra che  $L(M') = L$  per induzione su  $|y| \geq 0$  che  $\delta'([x], y) = [xy]$

**TEOREMA 5.18.** *Per ogni linguaggio regolare  $L$  esiste un automa  $M$  con minimo numero di stati tale che  $L = L(M)$ , unico a meno di isomorfismo (ovvero rinomina di stati).*

dim: Consideriamo il DFA  $(M)$  che accetta  $L$ , sappiamo che la relazione  $R_M$  partiziona  $\Sigma^*$  in classi di equivalenza corrispondenti agli stati di  $M$ . Da  $(2 \rightarrow 3)$  sappiamo che  $R_M$  raffina  $R_L$ , perciò  $M'$  (automa associato a  $R_L$  a seguito di  $(3 \rightarrow 1)$ ) ha un numero di stati  $\leq$  numero di stati di  $M$ .

Se ha un numero minore, allora  $M$  (considerato per ipotesi come il minimo) non sarebbe più il minimo  $\rightarrow$  contraddizione.

Se ha lo stesso numero di stati allora sono isomorfi, per dimostrare che sono isomorfi defisci  $f(q) = [x]$  sse  $\delta^\wedge(q_0, x) = q$  e deve essere **univoca** (biettiva).

Per essere univoca supponiamo che  $f(q) = [x]$  e  $f(q) = [y]$ , ma allora  $x R_M y$  (perché  $\delta^\wedge(q_0, x) = q$  e  $\delta^\wedge(q_0, y) = q$ ) e dato che  $R_M$  raffina  $R_L$  allora  $[x] = [y]$ .

f è suriettiva dato che esiste uno stato tale che la stringa finisce in esso e dato che  $|Q| = |Q'|$  allora è biettiva. === DA FINIRE ===

**TEOREMA 6.8.** *Sia  $G = \langle V, T, P, S \rangle$  una grammatica CF. Allora  $S \xrightarrow{G} \alpha$  se e solo se esiste un albero di derivazione con radice etichettata  $S$  per  $G$  che descrive  $\alpha$ .*

dim: faccio vedere che le foglie (da sx verso dx) dell'albero di derivazione contengono  $\alpha$

**TEOREMA 9.2.** *Se  $L$  è generato da una grammatica lineare destra, allora  $L$  è un linguaggio regolare.*

dim: (produzioni del tipo  $S \rightarrow \text{Terminale} \cup \text{Variabile}$  o  $\text{Terminale}$ )  
Costruisco un NFA che ha stati  $Q = V \cup T$ ,  $\Sigma = T$ ,  $S = q_0$ ,

$B \in \delta(A, a)$  sse  $A \rightarrow aB \in P$ ,  $\perp \in \delta(A, a)$  sse  $A \rightarrow a \in P$ ;  $\delta(\perp, a)$  è indefinito per ogni simbolo  $a \in \Sigma$ .

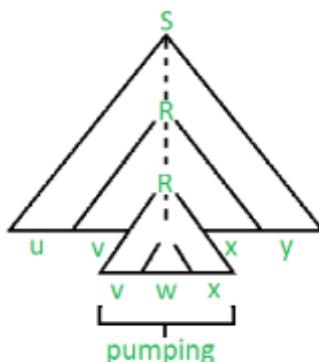
### PUMPING LEMMA (CF)

LEMMA 8.1 (Bar-Hillel, Perles, Shamir, 1961). *Sia  $L$  un linguaggio CF. Allora c'è una costante  $n \in \mathbb{N}$  (dipendente dal linguaggio  $L$ ) tale che per ogni  $z \in L, |z| \geq n$ , esistono stringhe  $uvwxy$  tali che*

- (1)  $z = uvwxy$ ,
- (2)  $|vx| \geq 1$ ,
- (3)  $|vwx| \leq n$ , e
- (4) per ogni  $i \geq 0$  vale che  $uv^iwx^i y \in L$ .

$$\exists n \in \mathbb{N} \forall z \left( \begin{array}{l} (z \in L \wedge |z| \geq n) \longrightarrow \exists u, v, w, x, y \\ \left( \begin{array}{l} z = uvwxy \wedge |vx| \geq 1 \wedge |vwx| \leq n \wedge \\ \forall i (i \in \mathbb{N} \rightarrow uv^iwx^i y \in L) \end{array} \right) \end{array} \right)$$

dim: se considero  $n = 2^k$  con  $k = |V|$  allora  $|z| \geq 2^k$  e per forza ci sarà almeno un cammino radice-foglia che contiene  $k + 1$  nodi (ricorda da algo. che un albero binario con  $2^k$  foglie ha altezza  $k + 1 \Rightarrow$  foglie =  $2^{\# \text{livelli (altezza)}}$ ) e se consideriamo che l'ultimo nodo foglia ha un'altra foglia allora in totale c'è un cammino con  $k + 2$  nodi al suo interno (piccionaia). Tutti i nodi escluso il  $k + 2$  esimo nodo contengono variabili, perciò per forza una variabile si ripete. Senza perdita di generalità, assumo che la grammatica sia in forma normale di Chomsky. Scompongo una stringa  $z$  appartenente al linguaggio come  $z = uvwxy$ .



Chiamiamo R la variabile ripetuta. Come si può vedere nello schema, siccome G è in CNF, e  $|n| \geq 2$  in P avremo una produzione del tipo:

$$R \rightarrow BC$$

Per costruzione, il cammino tra il primo vertice dall'alto in cui appare R (chiamiamolo  $v_1$ ) e le foglie ha lunghezza al più  $k + 1$ , quindi  $|vwx| \leq n$ . La seconda occorrenza di R apparirà o nel sottoalbero destro di  $v_1$ , o nel suo sottoalbero sinistro: di conseguenza, almeno uno dei due deve essere non vuoto, quindi  $|vx| \geq 1$ . Notiamo che in  $^*$  passi in G da R posso derivare  $S \Rightarrow uRy \wedge R \Rightarrow vRx \wedge R \Rightarrow w$ , quindi

$$R \Rightarrow vRx \Rightarrow vvRxx \Rightarrow vvvRxxx \Rightarrow \dots \Rightarrow v^i Rx^i \Rightarrow v^i wx^i.$$

Definitivamente,  $S \Rightarrow uv^i wx^i z \in L$ .

**TEOREMA 6.12.** *Ogni linguaggio CF non vuoto è generato da una grammatica CF priva di simboli inutili.*

dim: posso usare per dimostrare se una grammatica genera un linguaggio vuoto il principio di eliminazione dei terminali/variabili inutili, infatti se procedo a ritroso dai terminali aggiungendo ogni volta tutte le variabili che mi portano in questi terminali (considero le produzioni a ritroso) allora vedo se riesco a raggiungere S, se non lo raggiungo dunque il linguaggio generato è vuoto.

Per ridurre una grammatica in forma di Chomsky:

- elimino i simboli **inutili**: partendo dal basso (risalgo) elimino i nodi che non raggiungono i terminali. Successivamente partendo dall'alto (scendo) elimino i nodi non raggiunti da S.
- elimino le  **$\epsilon$ -produzioni**: "porto su"  $\epsilon$  alla variabile che la precede nella produzione ed elimino  $\epsilon$  dalla produzione corrente
- elimino le **produzioni unitarie**: collego tutte le variabili ai terminali (eliminando le produzioni intermedie)

**TEOREMA 6.16** (Chomsky, 1959). *Ogni linguaggio CF senza  $\epsilon$  è generato da una grammatica in cui tutte le produzioni sono della forma  $A \rightarrow BC$  e  $A \rightarrow a$ .*

dim: partendo da una grammatica senza produzioni unitarie ( $A \rightarrow B$ ), simboli inutili e  $\epsilon$ -produzioni, considero  $G = (V, T, P, S)$  e  $A \rightarrow X_1 X_2 \dots X_m \in P$  con  $m$  (lunghezza)  $> 1$ .

- Se  $m = 1$  allora apposto dato che non ci sono produzioni unitarie per ipotesi;
- Se  $m = 2$  e  $X_1, X_2 \in V$  allora apposto siamo nella forma cercata

- Se  $m \geq 2$  e per qualche  $X_i \in T$  allora per ogni  $X_i \in T$  introduco in  $V$  una nuova variabile  $B_i$  e aggiungo la produzione  $B_i \rightarrow X_i$ , rimpiazzando quella originale in  $A \rightarrow X_1 X_2 \dots X_m$
- Se  $m \geq 2$  e tutti gli  $X_i \in V$  allora rimpiazzo la produzione  $A \rightarrow X_1 X_2 \dots X_m$  con le produzioni  $A \rightarrow BX_3 \dots X_m$ ,  $B \rightarrow X_1 X_2$  con  $B$  nuova variabile aggiunta.

TEOREMA 6.20 (Greibach, 1965). *Ogni linguaggio CF senza  $\epsilon$  è generato da una grammatica in cui tutte le produzioni sono della forma  $A \rightarrow a\alpha$ , ove  $\alpha$  è una stringa (eventualmente vuota) di simboli non terminali.*

dim: non l'abbiamo fatta

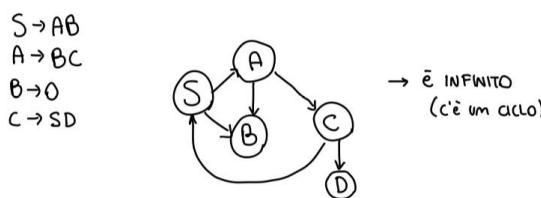
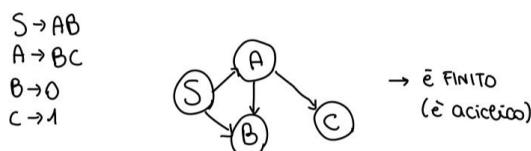
TEOREMA 8.7 (Vuoto, Finito, Infinito). *Data una grammatica CF  $G = \langle V, T, P, S \rangle$ , i problemi:*

- (1)  $L(G) = \emptyset$ ,
- (2)  $L(G)$  è finito,
- (3)  $L(G)$  è infinito,

*sono decidibili.*

dim:

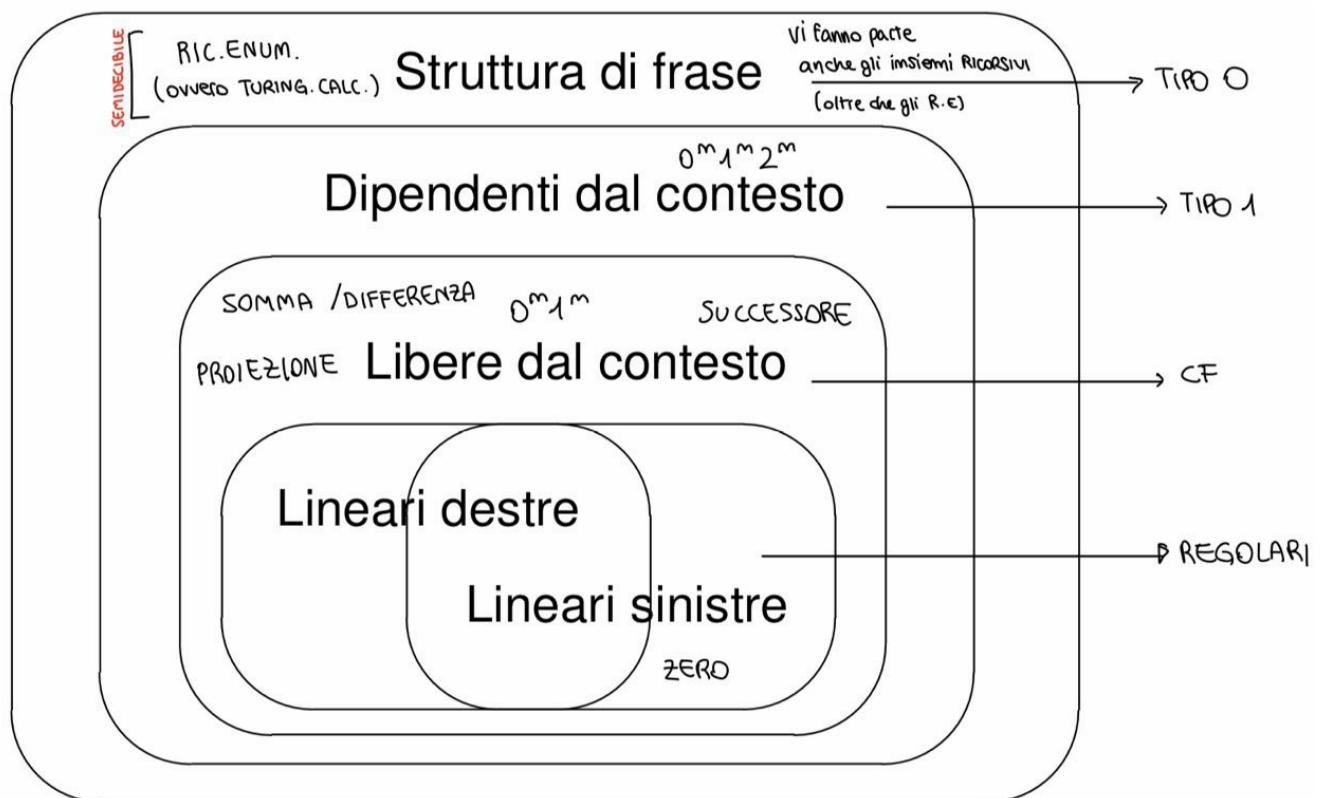
1. se durante l'eliminazione di simboli inutili vedo che da  $S$  non arrivo a terminali allora il linguaggio generato è vuoto;
2. se creo un grafo, che mi rappresenta le varie produzioni, dalla  $G$  in forma normale di Chomsky allora se questo è aciclico vuol dire che è finito;
3. stesso procedimento di (2), se è ciclico vuol dire che è infinito.



TEOREMA 8.8 (Appartenenza). *Data una grammatica CF  $G = \langle V, T, P, S \rangle$ , e una stringa  $z$ , il problema  $z \in L(G)$  è decidibile.*

dim: verificare per tutte le produzioni se arrivo a  $z$  e' semidecidibile, perciò parto con  $G$  in forma di Greibach ( $A \rightarrow aB_1B_2\dots B_n$ ) e se  $z$  ha una derivazione allora questa e' lunga esattamente  $|z|$ , perciò generiamo tutte le derivazioni di  $|z|$  passi e verifichiamo se una di queste deriva  $z$ .

Mentre l'equivalenza di due linguaggi CF e' indecidibile.



RICORSIVI(appartenenza decidibile): reg,CF, tipo 1 (CS)

RICORSIVI PARZIALI(appartenenza semidec.): tipo 0

NON DECIDIBILE(appartenenza non dec.): produttivi

## REQUISITI DI UN ALGORITMO

- Un algoritmo è di lunghezza finita
- Esiste un agente di calcolo che porta avanti il calcolo
- L'agente di calcolo ha a disposizione una memoria
- Il calcolo avviene per passi discreti
- Il calcolo non è probabilistico
- Non deve esserci alcun limite finito alla lunghezza dei dati in ingresso
- Non deve esserci alcun limite alla quantità di memoria disponibile
- Deve esserci un limite finito alla complessità delle istruzioni eseguibili dal dispositivo
- Sono ammesse esecuzioni con un numero di passi finito ma illimitato

DEFINIZIONE 11.1. Una *Macchina di Turing M* consiste di:

- (1) un alfabeto finito  $\Sigma = \{s_0, \dots, s_n\}$  con almeno due simboli distinti  $s_0 = \$$  (blank) e  $s_1 = 0$  (tally);
- (2) un insieme finito di stati  $Q = \{q_0, \dots, q_m\}$  tra i quali vi è lo stato iniziale  $q_0$  (dunque  $Q$  è non vuoto);
- (3) un insieme finito non vuoto di istruzioni (o quintuple)  $P = \{I_1, \dots, I_p\}$  ognuna delle quali di uno dei seguenti 2 tipi base:
  - $q \ s \ q' \ s' \ R$
  - $q \ s \ q' \ s' \ L$

tal che non esistono due istruzioni che iniziano con la medesima coppia  $q, s$ .

DEFINIZIONE 12.1. Si dicono *funzioni ricorsive di base* le seguenti funzioni:

- la funzione costante 0:  $\lambda x. 0$ ;
- La funzione successore S:  $\lambda x. x + 1$ ;
- La funzione identità, o i-esima proiezione,  $\pi_i : \lambda x_1 \dots x_n. x_i$  con  $1 \leq i \leq n$ .

DEFINIZIONE 12.2. Una funzione  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  si dice:

- definita per *composizione* da  $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$  e  $h : \mathbb{N}^k \rightarrow \mathbb{N}$  se  $f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$
- definita per *ricorsione primitiva* da  $g : \mathbb{N}^{n-1} \rightarrow \mathbb{N}$  e  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  se
$$\begin{cases} f(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1}) \\ f(x_1, \dots, x_{n-1}, y+1) = h(x_1, \dots, x_{n-1}, y, f(x_1, \dots, x_{n-1}, y)) \end{cases}$$

Tutte e sole le funzioni definibili a partire dalle precedenti funzioni ricorsive di base mediante composizione e ricorsione primitiva definiscono l'insieme delle funzioni *primitive ricorsive*. L'idea è quella di costruire via via funzioni effettiva-

Essendo funzioni totali allora sono definite per tutti gli argomenti (ovvero terminano sempre) ed e' una limitazione perché non tutte le funzioni calcolabili sono definite su tutti gli input.

dim: Per induz. strutturale sulla complessità (numero di operazioni di composizione e ricorsioni primitive). Le funzioni ricorsive di base sono totali perché hanno 0 ric e 0 comp.

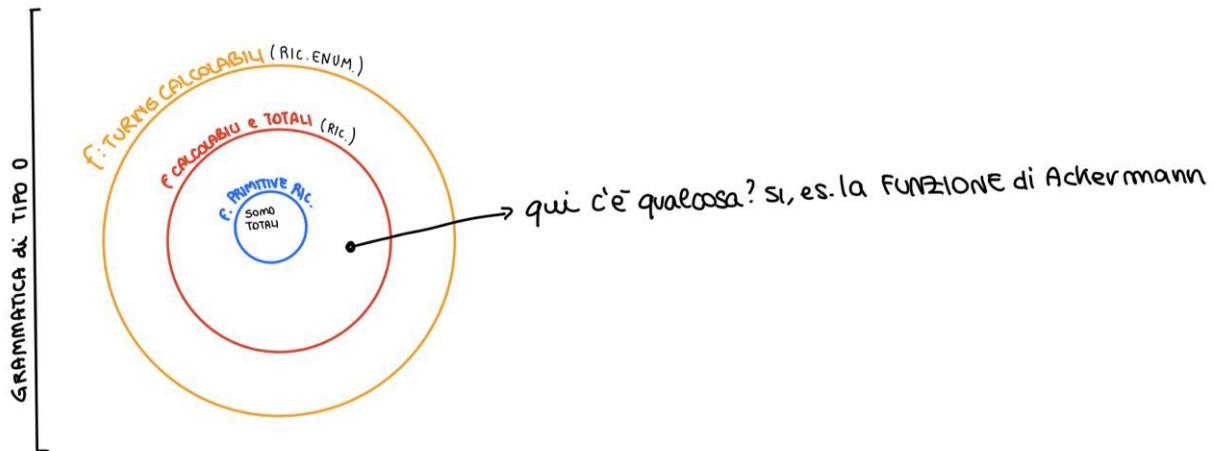
Se facciamo la composizione (assumendo per ipotesi che  $g_i$ ,  $h$  siano totali), allora anche la funzione  $f$  sara totale.

Se

**TEOREMA 12.8.** *Le funzioni primitive ricorsive sono totali.*

facciamo la ricorsione primitiva (assumendo per ipotesi che  $g$ ,  $h$  siano totali) allora anche la funzione  $f$  sara totale.

Esiste qualcosa tra funzioni primitive ricorsive e funzioni totali e calcolabili? si, la funzione di Ackermann e' un esempio.



**TEOREMA 12.9.** *ack non è ricorsiva primitiva [25].*

La dimostrazione del Teorema 12.9 è basata sul fatto che  $ack$  cresce più velocemente di qualunque funzione ricorsiva primitiva. Questa funzione, universalmente accettata come funzione calcolabile non è primitiva ricorsiva pur essendo totale.

dim: si dimostra che per qualsiasi funzione  $f$   $A$  maggiora  $f$  se esiste un  $t$  tale che per ogni  $n$ -pla di numeri  $x_1, \dots, x_n$  vale che  $f(x_1, \dots, x_n) < A(t, \max\{x_1, \dots, x_n\})$

dim (**terminazione**) : attraverso un piano cartesiano e' possibile vedere come da ogni passo al successivo una variabile (tra x e y) decrementi di 1, in modo da spostarsi (nel piano) verso sx o verso il basso e quando arrivo a x=0 ho terminato.

$$A(0, y) = y + 1$$

$$A(x + 1, 0) = A(x, 1)$$

$$A(x + 1, y + 1) = A(x, A(x + 1, y))$$

Dunque A e' calcolabile se esiste una MDT che la calcola e totale perché e' definita per tutti gli input.

**DEFINIZIONE 12.10.** Sia  $f$  una funzione totale  $n + 1$  aria, allora definiamo la funzione

$$\begin{aligned} \varphi(x_1, \dots, x_n) &= \mu z. (f(x_1, \dots, x_n, z) = 0) \\ &= \begin{cases} \text{il più piccolo } z \text{ t.c. } f(x_1, \dots, x_n, z) = 0 & \text{se } z \text{ esiste} \\ \uparrow & \text{altrimenti} \end{cases} \end{aligned}$$

Chiaramente la funzione  $\mu$ -operatore e' ricorsiva parziale per definizione e la classe di funzioni parziali ricorsive e' la minima classe di funzioni contenenti le funzioni primitive ricorsive e chiusa per  $\mu$ -ricorsione.  $\rightarrow$  funz. calc. tot. +  $\mu$ -op = funz. parz. ric.

**TEOREMA 12.14.**  $f : \mathbb{N} \rightarrow \mathbb{N}$  è parziale ricorsiva se e solo se è Turing-calcolabile.

dim:

( $\Rightarrow$ ) dimostriamo una tesi ancora più forte, ovvero se  $f$  e' ricorsiva allora esiste una mdT che la calcola con le seguenti ipotesi aggiuntive:

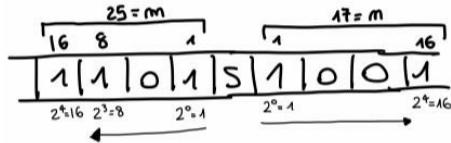
- Funziona anche se il nastro a sx e dx dell'input e' sporco (ovvero non \$)
- Quando termina, termina a dx dell'input che non viene modificato durante l'esecuzione, e l'output e' a dx della testina
- La parte che sta a sx dell'input non viene modificata.

Le funzioni di base hanno una mdT che la genera.

Per la composizione si sposta ogni volta l'input a dx dell'output calcolato dalla procedura precedente e si fa partire una nuova funzione ed infine sposto l'output a dx dell'input e mi posiziono tra di essi.

Per la ricorsione primitiva si implementa come se fosse uno stack sviluppando gli indici x e y.

( $\Leftarrow$ ) rappresentiamo una generica descrizione istantanea (ID) con una quadrupla  $ar(\alpha) = (q, s, m, n)$  con m e n come somme binarie dei valori a sx(m) e dx(n) della testina.



Definiamo inoltre 3 funzioni che rappresentano rispettivamente il cambio di stato, cambio di simbolo e cambio di direzione (di movimento)

$$\delta_q: Q \times \Sigma \rightarrow Q \quad \delta_\Sigma: Q \times \Sigma \rightarrow \Sigma \quad \delta_x: Q \times \Sigma \rightarrow \{0, 1\}$$

e quindi si possono descrivere le quadruple della transizione di stato attraverso queste funzioni di rappresentazione:

$$\text{prossimo stato: } g_q(q, s, m, n) = \delta_q(q, s)$$

$$\text{nuovo simbolo: } g_\Sigma(q, s, m, n) = (m \bmod 2)(1 - \delta_x(q, s)) + (n \bmod 2)\delta_x(q, s)$$

$$\text{nuova m che si forma: } (2m + \delta_\Sigma(q, s))\delta_x(q, s) + (m \bmod 2)(1 - \delta_x(q, s))$$

$$\text{nuova n che si forma: } (2n + \delta_\Sigma(q, s))(1 - \delta_x(q, s)) + (n \bmod 2)\delta_x(q, s)$$

che sono funzioni primitive ricorsive (fatte da composizione e ric. prim)

(la nuova m, ad es, se lo spostamento è verso sx, verrà dimezzata perché le tolgo una cifra binaria, mentre se lo spostamento è verso dx verrà raddoppiata perché le aggiungo una cifra binaria)

Rappresentiamo t transizioni con le funzioni  $P_Q, P_\Sigma, P_M, P_N$  e la terminazione l'ho quando  $P_Q(t, q_0, s_0, m_0, n_0) = k + 1$  che sarebbe il minimo t tale per cui la mdt termina nello stato  $k + 1$  (che non è definito).

$$\text{mt. } (k + 1 - P_Q(t, q_0, s_0, m_0, n_0) = 0).$$

\*\*\*\*DA TERMINARE\*\*\*

**Tesi di Church-Turing:** *La classe delle funzioni "intuitivamente calcolabili" coincide con la classe delle funzioni Turing calcolabili.*

spiegaz: per dimostrare che  $\text{Java} \geq \text{mdT}$ , costruisco una classe che mi sviluppa una  $\text{mdT}$ . Per dimostrare che  $\text{mdT} \geq \text{Java}$  ricordo che  $\text{Java} \rightarrow \text{Assembly} \rightarrow \text{RAM} \rightarrow \text{mdT}$  e  $\text{mdT} \geq \text{RAM}$ .

**Se dimostro che una funzione non è parziale ricorsiva allora in virtù della tesi di Church-Turing essa non è calcolabile in nessun modo effettivo.**

Se ho un algoritmo allora per forza esiste una funzione parziale ricorsiva che lo calcola.

**TEOREMA 14.4.** *Esistono funzioni (totali)  $f : \mathbb{N} \rightarrow \mathbb{N}$  non Turing calcolabili.*

dim:  $|\{f: N \rightarrow N\}| \geq |\{f: N \rightarrow \{0, 1\}\}| = 2^{|N|} > |N|$  che è il numero di funz. Turing calc. (enumerazione di Gödel)

Esiste una mdT universale che (fa parte della enumerazione) e data una coppia  $(x_1, x_2)$  mappata in  $y$  con la funzione pair, restituisce l'output della mdT di indice  $x_1$  che ha come parametro di input  $x_2$ .

## HALTING PROBLEM

**LEMMA 15.1.** *Non esiste una funzione ricorsiva  $g$  tale che per ogni  $x$ :*

$$g(x) = \begin{cases} 1 & \text{se } \varphi_x(x) \downarrow \\ 0 & \text{se } \varphi_x(x) \uparrow \end{cases}$$

dim: se indice  $i_0$  per assurdo esistesse una mdT di tale che  $g = \varphi_{i_0}$  allora definiamo la

$$g'(x) = \begin{cases} \uparrow & \text{se } g(x) = 1 = \varphi_{i_0}(x) \\ 0 & \text{se } g(x) = 0 = \varphi_{i_0}(x) \end{cases}$$

dunque anche  $g'$  sarà calcolabile ed esiste un indice  $i_1$  tale che  $g' = \varphi_{i_1}$ .

$$\begin{aligned}\varphi_{i_1}(i_1) \downarrow &\Leftrightarrow g'(i_1) = 0 \Leftrightarrow g(i_1) = 0 \Leftrightarrow \varphi_{i_1}(i_1) \uparrow \\ \varphi_{i_1}(i_1) \uparrow &\Leftrightarrow g'(i_1) \uparrow \Leftrightarrow g(i_1) = 1 \Leftrightarrow \varphi_{i_1}(i_1) \downarrow\end{aligned}$$

Assurdo.

$M_x(y) \downarrow$  e' semidecidibile ma non decidibile (per la tesi di C.Turing).

$\forall y M_x(y) \downarrow$ , ovvero stabilire se un programma e' totale, e' indecidibile.

**TEOREMA 14.7 (s-m-n).** Per ogni coppia di interi  $m, n \geq 1$  esiste una funzione ricorsiva totale  $s_n^m$  di  $m+1$  variabili tale che per ogni  $x, y_1, \dots, y_m$  abbiamo che:

$$\forall z_1 \dots z_n. \varphi_x(y_1, \dots, y_m, z_1, \dots, z_n) = \varphi_{s_n^m(x, y_1, \dots, y_m)}(z_1, \dots, z_n)$$

dim:  $\varphi_x$  e' la mdT di indice  $x$  che calcola la funzione  $f$ .

$\varphi_{s_n^m}$  con  $s_n^m(x, y_1, \dots, y_m)$  e' l'indice della mdT che assegna i parametri  $(x, y_1, \dots, y_m)$  (localmente) e poi richiama la funzione  $f$  della mdT  $\varphi_x$  con i parametri  $(z_1, \dots, z_n)$ .

Nella pratica poi io utilizzo s-m-n per **passare da una funz. ric. parz. ad una totale**, infatti  $\forall x \forall y (\psi(x, y) = \varphi_{g(x)}(y))$ .

Dato  $z$  (indice della mdT che calcola  $\psi(x, y)$ ), con la funzione  $g$  vado a cercare una mdT che assegna  $x$ , io so per certo che e' totale perche' esiste una mdT che assegna  $x \rightarrow$  questa mdT ha indice  $g(x)$  (RIC TOT).

Una volta trovata la mdT  $\varphi_{g(x)}$  gli passiamo  $y$  come parametro e questa puo anche non terminare.

Un insieme  $A$  e' **RE** se  $\exists y W_y = A$ , ovvero se esiste una funzione  $\psi$  tale che la mdT che la calcola ha come dominio  $A$ . Quindi io so che stabilire se un elemento  $\in A$  e' semidecidibile.  $A$  e' **RE COMPLETO** se  $K \leq A$

**TEOREMA 17.5 (Teorema di Post).** Un insieme  $A$  è ricorsivo se solo se  $A$  e'  $\bar{A}$  sono r.e.

dim: ( $\Rightarrow$ ) da  $A$  ricorsivo ho quindi la funzione caratteristica (ricorsiva totale) che mi dice se un elemento appartiene o no ad  $A$ .

Creo le funzioni semicaratteristiche assegnando  $\uparrow$  nel caso l'elemento non appartenesse ad  $A$  e così ho dimostrato che  $A$  e'  $\bar{A}$  sono re.

$$\psi(x) = \begin{cases} 1 & \text{se } f_A(x) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

( $\Leftarrow$ ) Se  $A$  e  $\neg A$  sono re allora hanno le loro funzioni semicaratteristiche e stabilire per ciascuna se  $z \in A \vee \neg A$  sarebbe semidecidibile, perciò si esegue a turno un'istruzione della mdT che calcola  $\psi_A$  e una della mdT che calcola  $\psi_{\neg A}$ , appena una termina si ferma la computazione.

$K$  e' RE,  $K$  non e' ricorsivo,  $\neg K$  non e' RE

TEOREMA 17.6 (Caratterizzazione degli insiemi r.e.). *Le seguenti affermazioni sono equivalenti:*

- (1)  $A \in \text{RE}$ ;
- (2) esiste  $\varphi \in \mathcal{PR}$ :  $A = \text{range}(\varphi)$ ;
- (3)  $A = \emptyset$  oppure esiste una funzione  $f \in \text{R}$  tale che  $A = \text{range}(f)$ .

(2) esiste  $g$  calcolabile (anche parziale) tale che  $A = \text{range}(g)$

(3)  $A = \emptyset$  oppure esiste una funzione  $f$  ricorsiva totale tale che  $A = \text{range}(f)$

dim:

(1  $\rightarrow$  2) Devo cercare una funzione che abbia il codominio =  $A$ .

Se  $A$  e' RE allora esiste una mdT  $M_x$  tale che il suo dominio coincida con  $A$ .

Creo una funzione ricorsiva  
abbia come output  $y$  se  $M_x(y) \downarrow$  e  
funzione desiderata.

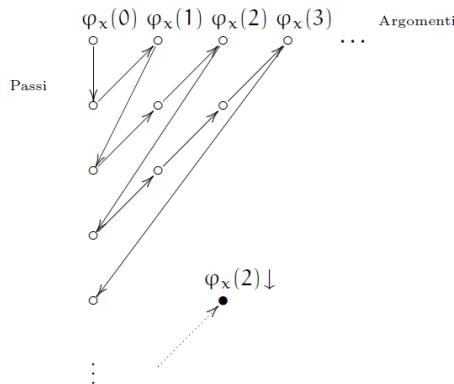
$$\delta(y) = \begin{cases} y & \text{se } \varphi_x(y) \downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

parziale che  
così ottengo la

(2  $\rightarrow$  3) se  $A = \emptyset$  allora il codominio di  $f$  è vuoto, ok.

se  $A \neq \emptyset$  devo dimostrare che  $g$  e  $f$  hanno lo stesso codominio,  
allora sia  $g$  calcolata da una mdT  $M_x$  e devo costruire una funzione totale che mi dica  
con quali output la macchina termina.

Utilizzo il **dovetail** (diagonalizzazione) e so che prima o poi termina, dato che  $A \neq \emptyset$  e  
 $\exists y \varphi_x(y) \downarrow$ .



Mi salvo quindi l'output della prima che termina e lo considero come primo elemento di  $f$ , successivamente definisco  $f$  come segue:

$$f \begin{cases} f(0) = y_0 \\ f(y+1) = \begin{cases} \varphi_x(z) & \text{se } \varphi_x(z) \downarrow \\ f(y) & \text{else} \end{cases} \end{cases}$$

output nuovo  
se  $\varphi_x(z) \downarrow$   
output vecchio

ogni volta che trovo un nuovo valore lo assegno alla  $y$  corrente, altrimenti setto il vecchio valore

facendo così,  $f$  è totale e  $A = \text{range}(f)$

(3 → 1) se  $A = \emptyset$  allora il codominio di  $f$  è vuoto e  $\emptyset$  sappiamo che è RE, ok.

se  $A \neq \emptyset$  allora ho un insieme ric. e devo dimostrare che è RE.

Devo definire una funzione semicaratteristica che mi dice se  $x \in A$

$i=0;$   
while ( $f(i) \neq x$ )  
     $i++;$   
return 1;

è ricorsiva parz.  
(può andare in loop)

TEOREMA 17.10. *Le seguenti affermazioni sono equivalenti:*

- (1)  $A$  è ricorsivo;
- (2)  $A = \emptyset$  oppure  $A = \text{range}(f)$  con  $f \in \mathcal{R}$  non decrescente.

dim:

(→)  $A$  è ricorsivo, se  $A = \emptyset$  ok

se  $A \neq \emptyset$  allora esiste una funzione caratteristica  $g$  che mi dice se  $a \in A$  (che però può essere decrescente).

Cerco il minimo valore  $a$  di  $A$ , tale per cui la funzione caratteristica  $f(a)$  ritorna 1 e assegno a  $f(0) = a$ , successivamente definisco  $f(n+1)$ :

$$f(n+1) = \begin{cases} n+1 & \text{se } n+1 \in A \\ f(n) & \text{altrimenti} \end{cases}$$

ed è chiaro che sia totale e non decrescente.

(←) se  $A = \emptyset$  ok

Se  $A$  è finito allora è già ricorsivo.

Se  $A$  è infinito e devo cercare se  $x \in A$ , con  $f$  funzione ric tot, lancio in cascata  $f(0), f(1), \dots, f(i) \dots$  finché non trovo:

- $x, \text{ok allora } x \in A$
- $y > x \text{ allora } x \notin A \text{ perché essendo } f \text{ non decrescente allora se trovo qualcosa maggiore di } x \text{ vuol dire che } x \text{ non c'e'}$ .

TEOREMA 19.7. *Sia  $A, B \subseteq \mathbb{N}$ .*

- (1)  $A \preceq B \Rightarrow \bar{A} \preceq \bar{B}$ ;
- (2)  $A \preceq B \text{ e } B \in \text{RE} \Rightarrow A \in \text{RE}$ .
- (3)  $A \preceq B \text{ e } B \text{ ricorsivo} \Rightarrow A \text{ ricorsivo}$ .

Le proprietà positive si ereditano da dx verso sx

Le proprietà negative si ereditano da sx verso dx

$K \text{ e } \neg K$  non sono in relazione, neanche  $N \text{ e } \emptyset$

$\preceq$  è riflessiva e transitiva, non totale, non simmetrica, non un ordine

LEMMA 19.6. (1) *Sia  $A$  un insieme.  $\mathbb{N} \preceq A$  se e solo se  $A \neq \emptyset$ .*  
 (2) *Sia  $A$  un insieme.  $\emptyset \preceq A$  se e solo se  $A \neq \mathbb{N}$ .*

dim:

- (1) Se  $A \neq \emptyset$ , mettiamo che  $A = \{a\}$ , allora esiste un  $f$  t.c. se  $x \in \mathbb{N}$  allora  $f(x) \in A$  e se noi consideriamo la funzione costante  $f(x) = a$  allora si verificano le condizioni di riducibilità.

$$\begin{array}{c} \text{VERO} \quad \text{VERO} \\ x \in \mathbb{N} \rightarrow f(x) = a \in A \quad \checkmark \\ \text{FALSO} \quad \text{FALSO} \\ x \notin \mathbb{N} \rightarrow f(x) = a \notin A \quad \checkmark \end{array} \quad \begin{array}{l} \rightarrow \text{VERIFICATO} \\ \downarrow \\ \text{falso implica tutto} \end{array}$$

↳ mai possibile

- (2) Procedimento analogo a (1) soltanto che definisco  $f(x) = b$  costante.

LEMMA 19.8.  *$B$  ricorsivo e  $A \neq \emptyset$  e  $A \neq \mathbb{N}$  implica  $B \preceq A$ .*

## PRIMO TEOREMA DI RICORSIONE

TEOREMA 18.1 (Teorema di ricorsione di Kleene). *Sia  $t \in \mathcal{R}$  una funzione ricorsiva (totale). Allora esiste  $n \in \mathbb{N}$  tale che  $\varphi_n = \varphi_{t(n)}$ .*

dim: sia  $u$  l'indice di una generica mdT  $\varphi_u$ . Definiamo la funzione parziale ricorsiva:

$$\psi(u, x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \text{se } \varphi_u(u) \downarrow \\ \uparrow & \text{altrimenti} \end{cases} \quad \begin{array}{l} \text{con out l'output della mdT di indice output} \\ \text{della mdT } \varphi_u(u). \end{array}$$

Ovviamente la funzione  $\Psi$  è calcolabile e con teorema SMN posso ottenere  $g$  ric. tot. t.c.  $\psi(u, x) = \varphi_{g(u)}(x)$  e questa funzione ( $g$ ) e' indipendente da  $t$  (che e' ric. tot. per ip).

Compongo  $t \circ g$  e a sua volta e' una funz. ric. tot. ed  $\exists$  una mdT che la calcola.

Sia quindi  $(t \circ g)(v)$  calcolata da una mdT di indice  $v \rightarrow \varphi_v(v) = (t \circ g)(v)$ .

Vista la genericità di  $u$ , possiamo sostituirlo con  $v$  nella definizione di  $\psi(u, x)$ :

$$\psi(v, x) = \varphi_{g(v)}(x) = \begin{cases} \varphi_{t(g(v))}(x) & \text{se } \varphi_v(v) \downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

e poiché  $(t \circ g)(v)$  e' totale allora  $\varphi_v(v)$  termina sempre e dunque

$$\varphi_{g(v)}(x) = \varphi_{t(g(v))}(x)$$

## SECONDO TEOREMA DI RICORSIONE

TEOREMA 18.7. *Sia  $f : \mathbb{N}^2 \rightarrow \mathbb{N} \in \mathcal{R}$  una funzione ricorsiva (totale). Allora esiste una funzione  $v : \mathbb{N} \rightarrow \mathbb{N} \in \mathcal{R}$  ricorsiva totale tale che:*

$$\forall y \in \mathbb{N} : \varphi_{f(v(y), y)} = \varphi_{v(y)}.$$

dim: (non necessaria)

Mostrare che esiste una funzione  $v$  tale che  $\forall x: W_{v(x)} = \{p_x\}$ .

Devo partire da una funzione di 3 variabili dato che con SMN ridurrò ad una di 2 var.

$$\chi_{f(a,b,c)} = \begin{cases} 0 & \text{se } c = p_b \\ \uparrow & \text{else} \end{cases} \quad \begin{array}{l} \text{l'iesimo numero primo} \\ \text{è CALCOLABILE} \end{array} \quad \begin{array}{l} \text{e per teo SMN, } \exists f \text{ RIC.TOT. t.c.} \\ \forall a \forall b \forall c \quad \chi_{f(a,b,c)} = \varphi_{f(a,b)}(c) \end{array}$$

$W_{f(a,b)} = \{p_b\}$  e per il secondo teo. ric.  $\exists v \varphi_{f(v(y), y)} = \varphi_{v(y)} \rightarrow$

$$W_{f(v(y), y)} = W_{v(y)} = \{p_y\}$$

TEOREMA 18.13 (Teorema di Rice). *Sia  $\Pi$  una proprietà estensionale.  $\Pi$  è ricorsiva sse è banale (ovvero  $\Pi = \emptyset$  oppure  $\Pi = \mathbb{N}$ ).*

dim:

( $\leftarrow$ ) non sapremo mai dire se un programma fa qualcosa, soltanto se è' un programma o no (banale), ovvero se compila o no.

( $\rightarrow$ )

Se  $\Pi$  e' ricorsiva allora sia g la sua funzione caratteristica:

$$g(x) = \begin{cases} 1 & \text{se } x \in \Pi \\ 0 & \text{se } x \notin \Pi \end{cases} \quad \begin{array}{l} \text{e supponiamo per assurdo che } \Pi \text{ non sia banale.} \\ \text{Allora esistono } a, b \text{ tali che } a \in \Pi, b \notin \Pi. \end{array}$$

Definiamo la funzione h come segue:

$$h(x) = \begin{cases} b & \text{se } g(x) = 1 \\ a & \text{se } g(x) = 0 \end{cases} \quad \begin{array}{l} \text{per forza } h \text{ e' ric.tot. (dato che lo e' } g) \text{ e per il primo} \\ \text{teo ric. } \exists n_0 \text{ tale che } \varphi_{n_0} = \varphi_{h(n_0)} \end{array}$$

Da questo arriviamo all'assurdo:

- se  $n_0 \in \Pi$  allora  $g(n_0) = 1$  e perciò  $h(n_0) = b$  ma  $b \notin \Pi$ . assurdo.
- se  $n_0 \notin \Pi$  allora  $g(n_0) = 0$  e perciò  $h(n_0) = a$  ma  $a \in \Pi$ . assurdo.

TEOREMA 18.21 (Rice e Shapiro). *Sia  $\Pi$  una proprietà estensionale. Se  $\Pi$  è r.e. allora per ogni  $i \in \mathbb{N}$*

$$i \in \Pi \text{ sse } \exists j \in \Pi. \varphi_j \subseteq \varphi_i \wedge W_j \text{ è finito} \quad (\Phi)$$

**Se ho un estensionale che contiene numeri e dentro questo non trovo una mdT con dominio finito, allora non e' RE.**

dim:  $\varphi_j$  e' una macchina che svolge le stesse funzioni di  $\varphi_i$  ma per meno valori (ha un dominio più piccolo di  $W_i$ )

Assumendo  $\Pi$  estensionale e RE.

( $\rightarrow$ ) Sia  $i \in \Pi$ , se  $W_i$  è finito allora prendo  $j = i$  ed ho finito.

se  $W_i$  è infinito ma  $j \notin \Pi$  ogni qualvolta che  $\varphi_j \subseteq \varphi_i$  e  $W_j$  è finito. Definisco g:

$$g(z, t) = \begin{cases} \varphi_i(t) & \text{se } M_z(z) \text{ non termina in } \leq t \text{ passi} \\ & \uparrow \text{può non terminare} \\ & \uparrow \text{mdT da indice } z \text{ con parametrazione } z, \text{ se non termina allora gorna } \varphi_i(t) \\ & \text{altrimenti} \end{cases} \quad \begin{array}{l} \text{e decidibile perché simulo} \\ \text{al max t passi} \end{array}$$

per SMN esiste  $s$  ric.tot. tale che  $g(z, t) = \varphi_{s(z)}(t)$  e  $\varphi_{s(z)}(t) \subseteq \varphi_i(t)$  perché calcolano le stesse cose (infatti nella definizione di  $g(z, t)$  si richiama  $\varphi_i(t)$ ).

Ora:

- se  $z \in K$  vuol dire che  $M_z(z) \downarrow$ , quindi  $W_{s(z)}$  è finito (perché puo avere al più t elementi) e  $\varphi_{s(z)} \subseteq \varphi_i$ , ma per ipotesi  $s(z) \notin \Pi$
- se  $z \notin K$  implica che  $\varphi_{s(z)} = \varphi_i$ , pertanto  $s(z) \in \Pi$  (perché  $M_z(z) \uparrow$  e quindi  $g(z, t) = \varphi_i$ )

( $\leftarrow$ ) Supponendo per assurdo che esiste un  $i$  t.c.  $\exists j \varphi_j \subseteq \varphi_i$  ed inoltre  $W_j$  è finito e  $j \in \Pi$  e  $i \notin \Pi$ . Definisco  $g$  come segue:

$$g(z, t) = \begin{cases} \varphi_i(t) & \text{se } t \in W_j \text{ oppure } z \in K \\ \uparrow & \text{altrimenti} \end{cases}$$

$\uparrow \text{ M}_j(t) \downarrow$        $\uparrow \text{ 2AG nella computazione}$   
 $\downarrow \text{ M}_z(z) \downarrow$

per SMN esiste  $s$  ric.tot. tale che  $g(z, t) = \varphi_{s(z)}(t)$

Ora:

- se  $z \in K$  vuol dire che  $\varphi_{s(z)} = \varphi_i$ . Per l'ipotesi su  $i$  e l'estensionalità di  $\Pi$  allora  $s(z) \notin \Pi$
- se  $z \notin K$  implica che  $\varphi_{s(z)|W_j} = \varphi_i$ , pertanto  $s(z) \in \Pi$  (perché è estensionalmente equivalente a  $\varphi_i$ )

**DEFINIZIONE 19.16.** Un insieme  $A \subseteq \mathbb{N}$  è detto *produttivo* se esiste una funzione (totale) ricorsiva  $f_A$ , detta funzione produttiva di  $A$ , tale che:

$$\forall x \in \mathbb{N}. (W_x \subseteq A \Rightarrow f_A(x) \in A \setminus W_x)$$

Un insieme  $A$  si dice produttivo se esiste una funzione detta produttiva che per ogni elemento  $a_i \in A$  mi calcola un elemento  $b$  che non appartiene al dominio della macchina di indice  $a_i$  ma appartiene ad  $A$  (ovviamente il dominio è contenuto in  $A$ ).  $A$  non può essere RE perché, se fosse  $A = W_{x_0}$  per un certo  $x_0$  allora la funzione produttiva  $f$  risulterebbe con codominio vuoto perché  $A = W_{x_0}$ . ( $A \setminus W_{x_0} = \emptyset$ ).

e perciò non esisterebbe la funzione produttiva.

Per mostrare che  $A$  è produttivo (quindi non re) riduco  $\neg K$  ad esso  $\neg K \leq A$

Un insieme  $A$  è detto *creativo* se  $A$  è r.e. ed il suo complemento è produttivo, ovvero se:

$$A \in \text{RE} \text{ e } \forall x \in \mathbb{N}. (W_x \subseteq \bar{A} \Rightarrow f_A(x) \in \bar{A} \setminus W_x)$$

TEOREMA 19.17. Sia  $A \subseteq \mathbb{N}$ .

- (1)  $A$  produttivo  $\Rightarrow A$  non r.e.
- (2)  $A$  creativo  $\Rightarrow A$  non ricorsivo

dim:

- (1) se  $A$  è produttivo allora  $W_x$  è r.e. perché è un dominio di una m.d.T (def. r.e.) e quindi con la funzione di produzione riesco sempre a trovare un nuovo elemento fuori dal dominio che rende impossibile trovare un'enumerazione. Perciò con qualunque RE non riuscirò mai a "racchiudere"  $A$ .
- (2) se  $A$  è creativo allora  $\neg A$  è produttivo e  $A$  è RE. Ma se  $\neg A$  è produttivo allora non è RE e perciò non vale il teorema (ricorsivo  $\rightarrow$  RE + RE)

TEOREMA 19.22. Sia  $A$  un insieme produttivo. Esiste  $n_0 \in \mathbb{N}$  tale che

$$W_{n_0} \subseteq A \wedge |W_{n_0}| = \omega$$

ovvero esiste un sottoinsieme r.e. e infinito di  $A$ .

Costruisco un insieme  $B$  che è RE, è contenuto in  $A$  ed è infinito (lo considero come unione di tutti i sottoinsieme r.e. finiti che posso trovare in  $A$ ).

dim: parto dal dominio vuoto, prendo una m.d.T con quel dominio e trovo  $f$ , poi ripeto.

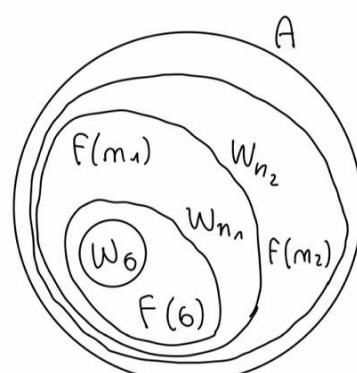
$A$  R.E. e sia  $F$  la sua funz. produttiva  
 $\emptyset$  è sicuramente  $\subseteq A$   $\xrightarrow{\text{indice } n_0} W_0 = \emptyset \subseteq A$  quindi  $f(0) \in A \setminus W_0$

Costruiamo m.d.T che termina solo in  $f(0)$   
calcolo l'indice  $\rightarrow m_1$   $\xrightarrow{f(m_0)} f(m_1)$

costruiamo m.d.T che termina solo in  $f(m_0) \in f(m_1)$   
di indice  $m_2$

e posso andare avanti all' $\infty$

$\Rightarrow$  trovo  $\infty$  punti di  $A$  produttivo



Trovo ogni volta un insieme  $W_n$  che ha cardinalità = cardinalità del precedente + 1, ovvero  $W_n = \{f(x_{n-1})\} \cup W_{n-1}$  con  $W_{n_0} = \emptyset$ .

Infine considero  $B$  come l'unione infinita di tutti questi insieme RE.

Esiste un modo unico per determinare  $B$  ma  $B$  non è unico ne esistono infiniti.

TEOREMA 19.20 (Myhill). *Sia  $A \subseteq \mathbb{N}$ .*

*$A$  r.e. completo sse  $A$  creativo*

dim:

( $\rightarrow$ ) se  $A$  è RE allora per ogni  $x$  r.e. :  $X \leq A$  e dato che  $K$  è completo e  $A$  è RE completo, allora  $K \leq A \rightarrow \neg K \leq \neg A$  e dunque  $\neg A$  è produttivo.

( $\leftarrow$ ) Se  $A$  è creativo allora è RE e  $\neg A$  è produttivo, quindi sia  $f$  funzione produttiva di

$$\psi(x, y, z) = \begin{cases} 0 & \text{se } y \in B \text{ e } z = f(x) \\ \uparrow & \text{altrimenti} \end{cases}$$

$\xrightarrow{\text{test R.E}}$   
 $\hookrightarrow \text{DECIDIBILE}$   
 $\text{test RICORSIVO}$   
 $\text{perché } f \text{ RIC.TOT}$

$\neg A$ . Sia  $B$  un generico insieme RE e definiamo

e dato che è calcolabile, per SMN esiste  $g$  ric. tot.  $\psi(x, y, z) = \varphi_{g(x, y)}(z)$

$$W_{g(x, y)} = \begin{cases} \{f(x)\} & \text{se } y \in B \\ \emptyset & \text{se } y \notin B \end{cases}$$

Per il secondo teorema di ricorsione esiste  $v$

ric. tot. tale che  $\forall y \in N \varphi_{g(v(y), y)} = \varphi_{v(y)}$

allora si ha che

e       $W_{g(v(y), y)} = W_{v(y)} = \begin{cases} \{f(v(y))\} & \text{se } y \in B \\ \emptyset & \text{se } y \notin B \end{cases}$

che

ora  
mostriamo  
 $f \circ g$  è la  
funzione di  
riduzione

da  $B$  ad  $A$  ( $B \leq A$ )

- se  $y \in B$  allora  $W_{v(y)} = \{f(v(y))\}$ .  $f(v(y)) \in A$  perché se non lo fosse allora vorrebbe dire che  $W_{v(y)} \subseteq \neg A$  ed essendo  $\neg A$  produttivo allora  $f(v(y))$  (funzione produttiva)  $\in \neg A \setminus W_{v(y)}$  che implicherebbe che  $f(v(y)) \notin W_{v(y)}$ . Assurdo
- se  $y \notin B$  allora  $W_{v(y)} = \emptyset$  che vorrebbe dire che  $W_{v(y)} \subseteq \neg A$  e pertanto  $f(v(y)) \in \neg A$

DEFINIZIONE 19.24.  $A \subseteq \mathbb{N}$  è *semplice* se

- $A$  è r.e.,
- $|\bar{A}| = \omega$ , ovvero  $\bar{A}$  è infinito,
- $\forall x \in \mathbb{N}. |W_x| = \omega \rightarrow A \cap W_x \neq \emptyset$ .

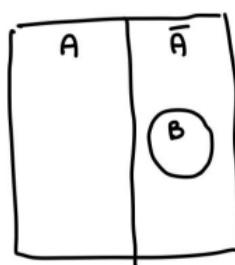
(3) per ogni  $B$  RE, se  $B$  è infinito allora  $A \cap B \neq \emptyset$

TEOREMA 19.25.

- (1)  $A$  semplice  $\rightarrow A$  non ricorsivo
- (2)  $A$  semplice  $\rightarrow A$  non creativo

dim:

- (1) Assurdo se  $A$  fosse ricorsivo allora anche  $\bar{A}$  sarebbe RE (per teo post) e dalle ipotesi di insieme semplice consideriamo  $B = \bar{A}$ . Allora anche  $B$  sarebbe RE infinito e dovrebbe risultare che  $A \cap (B = \bar{A}) \neq \emptyset$
- (2) Se  $A$  fosse creativo,  $\bar{A}$  sarebbe produttivo (per definizione). Quindi esisterebbe un  $B \subseteq \bar{A}$  tale che  $B$  è infinito e RE. Ma  $B \cap \bar{A} \neq \emptyset$  e' un assurdo dato che



TEOREMA 19.26. *Esiste un insieme semplice.*

$S \leq K$  ma  $K \neq S$

dim:

$S$  è r.e., quindi si riduce a  $K$  per la completezza di quest'ultimo. Al contrario, se  $K$  si riducesse ad  $S$  per il Teorema di Myhill  $S$  sarebbe creativo. Ma questo contraddice il teorema precedente.

Il numero di iterazione risulta limitato linearmente dalle dimensioni dell'input.

Dunque, dalla soluzione al problema decisionale si può ottenere la soluzione a quello funzionale con un numero di operazioni che dipende polinomialmente dall'algoritmo decisionale moltiplicato per le dimensioni dell'input.

**DEFINIZIONE 20.3.** Una k-MdT  $M$  è di tipo *decisionale* se ogni qual volta essa termina, raggiunge uno degli stati finali *yes* o *no*. L'output della macchina è, in questo caso, lo stato raggiunto.

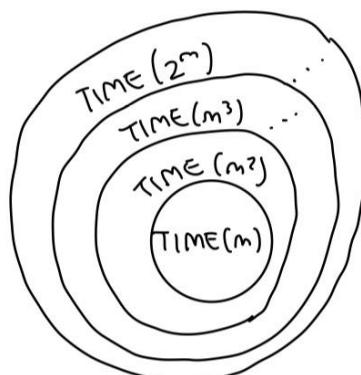
Una k-MdT  $M$  invece *calcola* una funzione se ogni qual volta termina, essa raggiunge lo stato finale  $h$ . In questo caso l'output è il contenuto del  $k$ -esimo nastro.

**DEFINIZIONE 20.5** (Classi in tempo). Un linguaggio  $L \subseteq \Sigma^*$  è *deciso* da una k-MdT  $M$  se per ogni  $x \in \Sigma^*$  vale che:

- Se  $x \in L$  allora  $M$  con input  $x$  termina nello stato *yes* (in breve  $M(x) = \text{yes}$ ).<sup>1</sup>
- Se  $x \notin L$  allora  $M$  con input  $x$  termina nello stato *no* (in breve  $M(x) = \text{no}$ ).

Se  $L \subseteq \Sigma^*$  è deciso da una k-MdT  $M$  e  $M$  opera in tempo  $f(n)$ , allora  $L \in \text{TIME}(f(n))$ .

$\text{TIME}(f(n))$  è una classe di complessità in tempo. Essa rappresenta l'insieme di linguaggi (o insiemi) che possono essere decisi in tempo limitato da una funzione. **Non dipende dal numero  $k$  di nastri**, se è polinomiale con 2 nastri lo sarà anche con uno.



TEOREMA 20.7. *Se  $M$  è una  $k$ -MdT che opera in tempo  $f(n)$ , possiamo costruire una 1-MdT  $M'$  equivalente e che opera in tempo  $O(nf(n) + f(n)^2)$ .*

dim:

L'idea è quella di simulare una  $k$ -MdT con una 1-MdT.

Simulo la situazione iniziale ponendo gli input iniziali dei  $k$  nastri concatenati all'inizio della 1-MdT, separati da un simbolo speciale che aggiunto all'alfabeto. Inoltre, all'alfabeto aggiungo un simbolo per ogni combinazione simbolo-stato dei  $k-1$  nastri aggiuntivi. Poi devo riscrivere  $M'$  in maniera tale da mimare, con più mosse, le mosse che eseguiva  $M$ . Ogni volta che devo aggiungere un simbolo alla fine dell'input di quello che prima era un singolo nastro, però, devo shiftare tutti i caratteri successivi a destra di una posizione. Questa è la causa dell'aumento del costo temporale a quadratico.

Conseguenze "informali":

1. **Non c'è una sostanziale differenza tra le 1-MdT e le  $k$ -MdT**
2. L'addendo  $nf(n)$  di solito "sparisce" nella complessità in quanto molto inferiore rispetto a  $f(n)^2$

TEOREMA 20.9 (Speed-up). *Sia  $L \in \text{TIME}(f(n))$ . Allora:*

$$\forall \epsilon > 0 : L \in \text{TIME} \left( \epsilon f(n) + n + 2 + \frac{\epsilon}{6} n \right).$$

dim:

Innanzitutto notiamo che i termini  $n + 2 + (\epsilon/6)n$  sono relativi al processamento iniziale dell'input, ma vengono spesso "cancellati" dal termine  $f(n)$ . L'intenzione è quella di codificare sequenze di simboli di lunghezza fissa  $m$  come simboli in una "base maggiore" (ad esempio se avessi la sequenza di simboli 10011111 in  $\Sigma = \{0,1\}$ , potrei convertirla in simboli esadecimali come 9F. In tal modo otterrei uno Speed-Up di fattore 4 in quanto ogni quadrupla di simboli viene valutata in un singolo passaggio (in realtà viene simulata in 6 passi, ma abbiamo saltato quella parte della dimostrazione)). Scegliendo  $m$  in funzione di  $\epsilon$  ( $m > \epsilon/6$ ) si ottiene il risultato.

applicazione pratica:

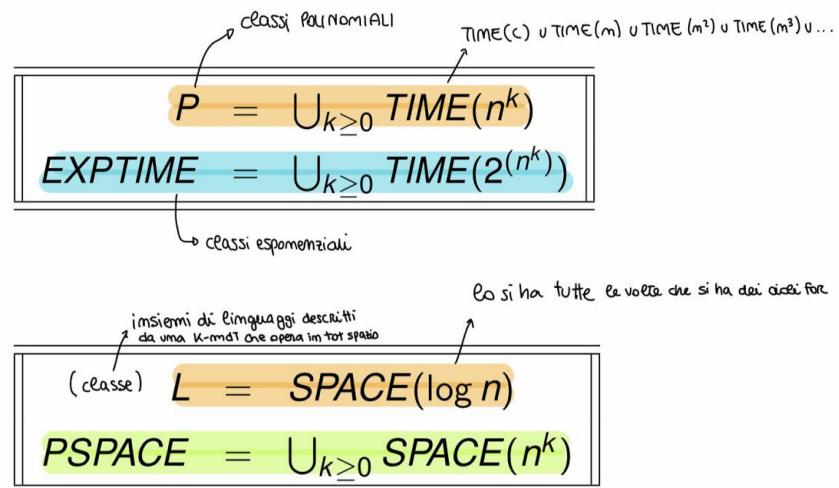
Passando da computer a 8 bit a computer a 64 bit, ottengo in una singola istruzione la computazione di 8 istruzioni nell'architettura precedente: **un miglioramento hardware può permettere sono un incremento lineare delle performance temporali**. Al contrario, algoritmi inefficienti rimangono tali anche se il calcolatore diventa 10, 100, 1000 volte più veloce. Si può dire in modo informale che questo teorema formalizza la notazione di  $O(f(n))$  per le MdT.

TEOREMA 20.11. *Sia  $L \in \text{TIME}(f(n))$ . Allora esiste un programma RAM che calcola la funzione caratteristica di  $L$  in tempo  $O(n + f(n))$ .*

*Sia  $P$  un programma RAM che calcola la funzione  $\phi$  in tempo  $f(n) \geq n$ . Allora esiste una 7-MdT che calcola  $\phi$  in tempo  $O(f(n)^3)$ .*

spiegazione:

**RAM** è un modello di calcolo ad **accesso diretto** in memoria (una specie di Assembler elementare). Per simularlo, si usa una MdT con tanti nastri quanti i registri (che hanno una funzione simile a quelli dell'Assembler, per l'appunto) utilizzati dalle istruzioni RAM, nello specifico, 7.



opera in **tempo** ( $\in \text{TIME}(\dots)$ ) e si guarda il numero di **passi**  
 opera in **spazio** ( $\in \text{SPACE}(\dots)$ ) e si guarda il numero di **celle**

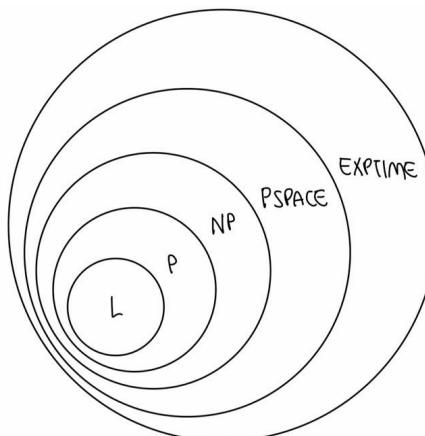
$\text{TIME}(f(n)) \subseteq \text{SPACE}(O(f(n)))$

dim:

$A \in \text{TIME}(f(n))$ .

In  $f(n)$  passi uso al massimo #nastri \* #passi per un nastro  $\rightarrow (k - 2) * f(|n|)$  celle  
 (il fattore  $k-2$  è dovuto al fatto che non conto lo spazio usato sul primo e l'ultimo nastro in scrittura).

corollario:  $P \subseteq \text{PSPACE}$



$P$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{TIME}(n^k)$
$\text{EXPTIME}$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{TIME}(2^{n^k})$
$\text{NP}$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{NTIME}(n^k)$
$\text{NEXPTIME}$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{NTIME}(2^{n^k})$
$L$	$\stackrel{\text{def}}{=} \text{SPACE}(\log n)$
$\text{PSPACE}$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{SPACE}(n^k)$
$\text{NL}$	$\stackrel{\text{def}}{=} \text{NSPACE}(\log n)$
$\text{NPSPACE}$	$\stackrel{\text{def}}{=} \bigcup_{k \geq 0} \text{NSPACE}(n^k)$

TEOREMA 20.29.  $\text{SPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))})$ .

dim:

Sia  $M$  la I/O-k-MdT che decide  $L$  in spazio  $\text{SPACE}(f(n))$ . Non si potrà mai trovare due volte nella stessa configurazione nastro-stato attuale perché altrimenti il determinismo garantirebbe la sua non terminazione (assurdo in quanto  $L$  è deciso da  $M$ ). Il numero di configurazioni diverse (e quindi di possibili passi computazionali) è dell'ordine:

$$\underbrace{(Q+3)}_{\text{stati}} \underbrace{\Sigma^{O(f(|x|))}}_{\text{stringhe sui nastri}} \underbrace{(|x|+1)f(|x|)^k}_{\text{posizione testina}}$$

So che in ogni nastro ci sono  $\Sigma^{O(f(|x|))}$  scelte e perciò questo è il costo influente.

corollario  $L \subseteq P$ :

$A \in L \rightarrow \exists \text{ I/O-k-MdT } M \text{ che decide } L \text{ in spazio } \log(n)$ .

$$A \in \text{SPACE}(\log(n)) \Rightarrow A \in \text{TIME}(2^{O(\log n)}) = \text{TIME}(2^{O(n)}) = P$$

DEFINIZIONE 20.14. Una ND-MdT  $M$  opera in tempo  $f(n)$  se, per ogni  $x \in \Sigma^*$ , ogni computazione non deterministica sull'input  $x$  ha al più lunghezza  $f(|x|)$ .

dove  $f(|x|)$  è l'altezza dell'albero di computazione

DEFINIZIONE 20.15. Se un linguaggio  $L \in \Sigma^*$  è deciso da una ND-MdT  $M$  che opera in tempo  $f(n)$ , allora  $L \in \text{NTIME}(f(n))$ .

lingue  
 insieme dei problemi che sono decisi da una mdt non det.  
 che opera in tempo polinomiale

$$\boxed{
 \begin{aligned}
 NP &= \bigcup_{k \geq 0} \text{NTIME}(n^k) \\
 \text{NEXPTIME} &= \bigcup_{k \geq 0} \text{NTIME}(2^{(n^k)})
 \end{aligned}
 }$$

**P**: insieme dei problemi che ammettono algoritmo **deterministico** che opera in tempo polin.

**NP**: insieme dei problemi che ammettono algoritmo **non-deterministico** che opera in tempo polin.

LEMMA 20.16.  $P \subseteq NP$ .

dim:

$A \in P \rightarrow \exists$  una 1-MdT che decide  $A$  in tempo  $O(n^h)$  per qualche  $h$ . Questa stessa MdT è anche una particolare ND-MdT dove l'albero di computazione degenera in una linea retta, quindi  $A \in NP$ .

TEOREMA 20.17.  $NTIME(f(n)) \subseteq \bigcup_{c \geq 1} TIME(c^{f(n)})$ .

dim:

Sia  $L \in NTIME(f(n))$  e  $M$  una ND-MdT che decide  $L$ . Sia  $c$  il massimo grado di non determinismo di  $M$ , ovvero la "divisione" massima di figli che compare nell'albero.

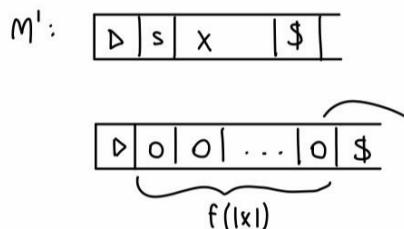
$$c = \max_{(q,s) \in Q \times \Sigma} |\{(q,s,q',s',M) \in P\}|.$$

Si può simulare  $M$  con un algoritmo deterministico che attraversa l'albero di computazione di  $M$  (ad esempio usando la ricorsione): l'attraversamento dell'albero diventa praticamente una singola visita DFS, che opera in tempo lineare rispetto al numero di nodi dell'albero. Il numero di nodi di un albero di altezza  $f(|x|)$  (costo della singola computazione deterministica) di grado  $c$  (numero massimo di computazioni parallele iniziata in un singolo nodo) a sua volta è  $O(c^{f(|x|)})$ , per cui il costo assume valore esponenziale.

approfondimento:

Per simulare una ND-MdT  $M$  con una MdT  $M'$  (soluzione ricorrente anche nell'implementazione di linguaggi ad alto livello) utilizzo una (almeno) 2-MdT in cui il secondo nastro contiene  $f(|x|)$  simboli compresi tra 0 e  $c-1$ . Questi simboli vengono incrementati quando raggiungo uno stato "no" per tenere traccia delle scelte effettuate in precedenza (da evitare per non finire in loop infiniti). Il secondo nastro verrà quindi inizializzato con  $f(|x|)$  zeri e rappresenterà un numero in base  $c$  che verrà incrementato di un'unità a ogni discesa che non termina nello stato "yes". La differenza sostanziale con un normale algoritmo di backtracking è che questa procedura attraversa un cammino radice-foglia a ogni "iterazione" (quindi il costo computazionale è molto alto), tuttavia può essere realizzato con una semplice MdT senza ulteriori strutture dati (come sarebbe ad esempio in C).

→ nel primo nastro eseguo la computazione come faceva la ND-mdT (quindi scende) e il secondo nastro lo uso per scegliere la regola di riscrittura (ovvero la direzione). Arrivo in fondo, se termine con YES allora ok, senno incremento di 1 il secondo nastro e ripeto la computazione dall'inizio.



corollario:

$$\begin{aligned} \text{NP} &\subseteq \text{EXPTIME} \\ \text{NP} &\subseteq \text{NPSPACE} = \text{PSPACE} \\ \text{PSPACE} &\subseteq \text{EXPTIME} \end{aligned}$$

**RIDUZIONI:** poter dire che se  $A \leq B$  e  $B$  è polinomiale allora  $A$  è polinomiale.

**DEFINIZIONE 21.1.** Dati due linguaggi  $L_1$  e  $L_2$  si dice che  $L_1 \subseteq \Sigma_1^*$  è riducibile a  $L_2 \subseteq \Sigma_2^*$  (in breve,  $L_1 \preceq L_2$ ) se esiste una funzione  $f : \Sigma_1^* \rightarrow \Sigma_2^*$  tale che:

- $f$  è calcolabile da una I/O-k-MdT (deterministica) in spazio  $O(\log n)$  e
- per ogni  $x \in \Sigma_1^*$  si ha che:

$$x \in L_1 \text{ sse } f(x) \in L_2$$

$f$  è chiamata una *riduzione* da  $L_1$  a  $L_2$ .

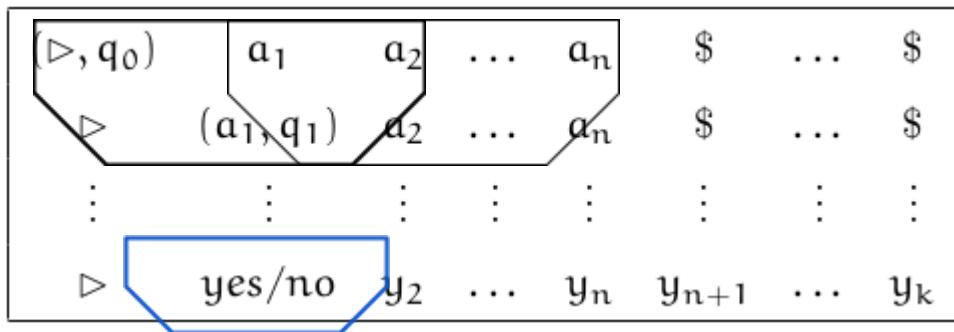
**DEFINIZIONE 21.2.** Data una classe  $\mathcal{C}$ , un problema  $L \in \mathcal{C}$  è  $\mathcal{C}$ -completo se ogni problema della classe può essere ridotto a  $L$ .

**DEFINIZIONE 21.7 (CIRCUIT VALUE).** **Input:** Un circuito logico con porte **and**, **or**, **e not** con una sola uscita e i valori fissati per gli ingressi.  
**Problema:** Stabilire se l'output è **true**.

**TEOREMA 21.9.** *CIRCUIT VALUE* è P-completo.

dim:

- 1) CIRCUIT VALUE  $\in P$ :  
è sufficiente percorrere il circuito (aciclico) con i dati in input valutando l'output di ogni porta in funzione dei suoi input. Ciò può essere ovviamente fatto in tempo polinomiale sulla dimensione del circuito.
- 2)  $P \leq CIRCUIT VALUE$ :  
Se un problema  $L$  sta in  $P$  esiste una MdT  $M$  a un nastro che opera in tempo  $n^k$  per qualche  $k$  che lo decide. Si può vedere la computazione di  $M$  come una matrice  $(n^k) \times (n^k)$  in cui ogni riga  $i$  rappresenta la configurazione del nastro dopo l' $i$ -esimo passo di  $M$ . In particolare, in corrispondenza della posizione della testina viene posta la coppia <simbolo letto - stato>. Se  $M$  termina in meno di  $n^k$  passi, si ricopia l'ultima riga fino al riempimento della matrice.



A partire da una codifica binaria dei simboli di  $\Sigma$  e di  $\Sigma \times Q$  è facile scrivere le funzioni booleane che calcolano la codifica del contenuto della cella  $M_{i,j}$ .

Il valore di un simbolo dipende solo dal simbolo soprastante e dai suoi adiacenti e se nessuno dei tre contiene la testina il valore sarà preservato.

Con  $\log_2(|\Sigma| + |Q| \times |\Sigma|) = h$  bit è possibile codificare in binario il contenuto di una cella; è facile quindi scrivere le funzioni booleane che calcolano la codifica del contenuto della cella  $M_{i,j}$ . Avrò quindi  $K$  circuiti ad 1 bit che verranno "traslati" lungo la matrice.

Riesco quindi a "trasformare" il problema iniziale in un circuito con  $h \times n^k$  ingressi. La decisione di una stringa appartenente a  $L$  equivale quindi a "eseguire" il circuito e a vedere quale valore ha il bit di output dell'ultima porta logica (quella in blu).

**DEFINIZIONE 21.8 (CIRCUIT SAT).** **Input:** Un circuito logico con porte **and**, **or**, e **not** con una sola uscita e  $n$  porte di ingresso.

**Problema:** Stabilire se esiste un assegnamento per le porte di ingresso che rende l'output **true**.

**TEOREMA 21.10.** **CIRCUIT SAT** è NP-completo.

dim:

1) CIRCUIT SAT  $\in$  NP:

Se ho una MdT  $M$  che risolve CIRCUIT VALUE, posso costruire una ND-MdT  $M'$  che sfrutta il non-determinismo per avviare una serie di computazioni deterministiche con  $M$  su tutti i possibili valori di input.

2) NP  $\leq$  CIRCUIT SAT:

Sulla base della dimostrazione precedente, e assumendo per semplicità che il massimo grado di non determinismo del linguaggio che si vuole decidere è 2, aggiungo ai circuiti costruiti sulla matrice un flag che sceglie quale dei 2 possibili input utilizzare. Gli input corrispondono alle 2 scelte non deterministiche che può fare la ND-MdT che decide il linguaggio in questione. Modificando i flag si può modellare l'albero di computazione della Macchina di Turing non deterministica. Di conseguenza, è possibile risolvere qualsiasi problema NP trasformandolo in un'istanza di SAT, quindi SAT è NP-completo.

NB: i due precedenti sono i teoremi di Cook e Levin!

Dato un insieme di variabili bool e una formula  $f$  (in FNC), il problema **SAT** e' quello di stabilire l'esistenza di un assegnamento di valori V/F alle variabili in grado di rendere la formula  $f$  vera.

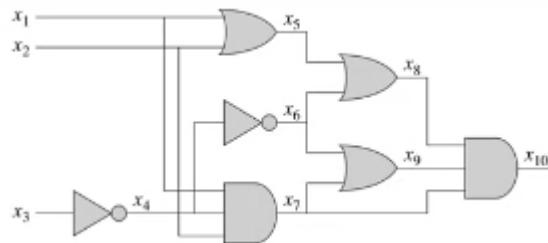
**TEOREMA 21.11.** *SAT è NP-completo.*

dim:

1)  $SAT \in NP$ :

La dimostrazione è analoga a quella di **CIRCUIT SAT**  $\in NP$ .

2)  $CIRCUIT SAT \leq SAT$ :



Guardando l'esempio in figura. Il circuito può essere codificato come

$$x_4 \leftrightarrow \neg x_3$$

$$x_5 \leftrightarrow x_1 \vee x_2$$

$$x_6 \leftrightarrow \neg x_4$$

$$x_7 \leftrightarrow x_1 \wedge x_2 \wedge x_4$$

$$x_8 \leftrightarrow x_5 \vee x_6$$

$$x_9 \leftrightarrow x_6 \vee x_7$$

$$x_{10} \leftrightarrow x_7 \wedge x_8 \wedge x_9$$

La formula di questa istanza SAT sarà dunque  $x_{10} \wedge \dots$  (l'output deve essere 1, e dopo la congiunzione bisogna inserire tutte le relazioni prima definite per codificare il circuito). La funzione di riduzione sarà la funzione che codifica il circuito e trasforma la formula calcolata in Forma Normale Congiuntiva (ad es.  $(l_1 \vee \dots \vee l_k) \wedge \dots \wedge (l_m \vee \dots \vee l_n)$  con  $1 < k < m < n$  e  $n$  numero di variabili in input).

risultato utile:

Per dimostrare che  $A$  è NP completo devo necessariamente dimostrare che  $A \in NP$  e che preso  $B$  che sappiamo essere NP completo,  $B \leq A$ .

Per facilitare ciò, per il primo punto possiamo utilizzare la caratterizzazione di NP come **Guess & Verify**: se è intuitivo che esiste un algoritmo deterministico polinomiale che può verificare la correttezza un certificato (detto anche testimone) per l'input  $x$ , ma la generazione del certificato richiede tempo polinomiale con una macchina non deterministica, allora  $A \in NP$ .

Per dimostrare che  $A$  è NP-completo (proprietà **NP-hardness**) è necessario ridurre un problema noto come NP completo ad  $A$  e per facilitare questo "basta" prendere in

considerazione una lista di problemi già noti come NP-completi e non necessariamente CIRCUIT SAT.

TEOREMA 21.14. **3SAT è NP-completo.**

dim:

1) 3SAT appartiene a NP:

Banale in quanto qualsiasi istanza di 3SAT è anche istanza di SAT.

2) SAT  $\leq$  3SAT:

Sia  $x$  input di SAT, che descrive una formula logica  $\Phi$ . Traduciamo ogni clausola  $l_1 \vee \dots \vee l_m$  di  $x$  in una equisoddisfacibile clausola di 3SAT:

- Se  $m = 1$ : la trasformo in  $l_1 \vee l_1 \vee l_1$
- Se  $m = 2$ : la trasformo in  $l_1 \vee l_2 \vee l_2$
- Se  $m = 3$  ho già un'istanza 3SAT valida
- Se  $m > 3$ : introduco una nuova variabile  $X$  e diviso la disgiunzione in  $l_1 \vee l_2 \vee X$  e applico la riscrittura alla clausola  $\neg X \vee l_3 \vee \dots \vee l_m$

Non è difficile convincersi che qualsiasi istanza  $yes$  di SAT viene trasformata in un'istanza  $yes$  di 3SAT in tempo deterministico polinomiale. La formula formula logica  $\Phi'$  così ottenuta non è infatti equivalente alla formula dell'input (può avere molte più variabili), ma è soddisfacibile se e solo se  $\Phi$  lo è.

TEOREMA 21.16. **NAESAT è NP-completo.**

dim:

1) NAESAT appartiene a P:

Sfrutto la proprietà Guess & Verify.

2) NP-Hardness:

Per dimostrare che NAESAT è NP-completo è sufficiente modificare la riduzione da CIRCUIT SAT a SAT. Per le leggi di De Morgan possiamo concentrarci solo su porte or e not. L'idea è di utilizzare una variabile aggiuntiva  $Z$ .

$$Y \leftrightarrow A \vee B \text{ diventa } (\neg Y \vee A \vee B) \wedge (\neg A \vee Y \vee Z) \wedge (\neg B \vee Y \vee Z)$$

$$Y \leftrightarrow \neg A \text{ diventa } (\neg Y \vee \neg A \vee Z) \wedge (Y \vee A \vee Z)$$

Sia  $\sigma$  che soddisfa CIRCUIT SAT. Consideriamo la traduzione non estesa con  $Z$ . Supponiamo, per assurdo, che nella prima clausola dell'ora tutti i tre letterali siano *true*. Allora la seconda e la terza clausola non sarebbero soddisfatte. Dunque  $\sigma$  non rende veri tutti e tre i letterali delle prime clausole dell'or. Ma allora con  $\sigma' = \sigma \circ [Z/false]$  (ossia assegno a  $Z$  il valore *false*) si ha una soluzione per NAESAT.

Viceversa, sia  $\sigma$  una soluzione per NAESAT. È immediato verificare che  $\sigma$  è una soluzione se e solo se  $\neg\sigma$  lo è. In una delle due  $Z$  sarà *false*: quella sarà la soluzione per CIRCUIT SAT.

NB: 2SAT è polinomiale. Infatti se ho solo clausole di 2 elementi, posso dedurre una serie di implicazioni dalla formula.

$$\text{es: } (A \vee \neg B) \wedge (\neg A \vee B) \wedge (A \vee B)$$

$$A \vee \neg B = \neg A \rightarrow B = B \rightarrow A$$

$$\neg A \vee B = A \rightarrow B = \neg B \rightarrow A$$

$$A \vee B = \neg A \rightarrow B = \neg B \rightarrow A$$

Posso disegnare un grafo orientato  $G = \langle V, E \rangle$ , in cui

- $V$  = l'insieme dei letterali che appaiono nelle implicazioni
- $E$  = l'insieme coppie ordinate di letterali  $(A, B)$  tali che tra le implicazioni ho  $A \rightarrow B$

Per verificare che esista una soluzione a una specifica istanza di 2SAT è dunque sufficiente verificare che questo grafo non compaiono cicli in cui sono presenti, in qualsiasi ordine, una variabile e il suo opposto. Esiste ovviamente un tale algoritmo che opera in tempo deterministico polinomiale.