

Algoritmi, macchine, grammatiche - Terza parte:
Grammatiche formali e automi



Carattere generativo del linguaggio:

(*) *Le cinquantasei lepri che ieri aggredirono il vescovo di Salerno sono fuggite in autobus verso Savona*

(**) *Uno lepre cinquantasei ieri aggredendo Salerno verso fuggire autobus autobus*

Che cos'è una grammatica formale?

alfabeto A : insieme finito di simboli

stringa (o **parola**) su A è una n -pla ordinata di elementi A

Es. alfabeto $A = \{a, b, c, d, e, +, -, *, /, (,)\}$

una stringa su A :

$(a, d, +, (, (, a, *)$

$a d + ((a *$

Altre stringhe su Al :

(i) $a + b$

(ii) $((+(- (*))))$

(iii) $a b c (d) /$

(iv) $a * (c - e)$

(v) $(e + c) / (a + (a - b))$

(vi) $+ a b$

Linguaggio L con alfabeto A : un sottoinsieme di tutte le stringhe su A

Stringhe grammaticali di L : le stringhe che appartengono a L

Una **grammatica** per un linguaggio L consente di specificare (in modo finito)
quali sono le stringhe grammaticali di L

Un esempio:

alfabeto $S = \{il, un, topo, gatto, rincorre, mangia, dorme, canta\}$

Una grammatica G su S :

- (1) $E \rightarrow SN SV$
- (2) $SN \rightarrow \text{Articolo Nome}$
- (3) $SV \rightarrow \text{VerbTrans SN}$
- (4) $SV \rightarrow \text{VerbIntr}$
- (5) $\text{VerbTrans} \rightarrow \text{rincorre}$
- (6) $\text{VerbTrans} \rightarrow \text{mangia}$
- (7) $\text{VerbIntr} \rightarrow \text{dorme}$
- (8) $\text{VerbIntr} \rightarrow \text{canta}$
- (9) $\text{Articolo} \rightarrow un$
- (10) $\text{Articolo} \rightarrow il$
- (11) $\text{Nome} \rightarrow topo$
- (12) $\text{Nome} \rightarrow gatto$

Produzioni, o regole di produzione, o regole di riscrittura:

le espressioni del tipo $e_1 \rightarrow e_2$

- **Simboli non terminali** - non fanno parte dell'alfabeto

Nel caso di G i simboli non terminali sono:

E, S, N, Articolo, Nome, VerbTrans, VerbIntr

- Simboli dell'alfabeto: **simboli terminali**

- **Assioma (o simbolo iniziale)**

L'assioma di G è E

Derivazione di *il gatto rincorre un topo* in *G*:

- | | |
|---|-----------------------------------|
| 1) E | assioma |
| 2) SN SV | da 1) per mezzo della regola (1) |
| 3) Articolo Nome SV | da 2) per mezzo della regola (2) |
| 4) <i>il</i> Nome SV | da 3) per mezzo della regola (10) |
| 5) <i>il gatto</i> SV | da 4) per mezzo della regola (12) |
| 6) <i>il gatto</i> Verb Trans SN | da 5) per mezzo della regola (3) |
| 7) <i>il gatto rincorre</i> SN | da 6) per mezzo della regola (5) |
| 8) <i>il gatto rincorre</i> Articolo Nome | da 7) per mezzo della regola (2) |
| 9) <i>il gatto rincorre un</i> Nome | da 8) per mezzo della regola (9) |
| 10) <i>il gatto rincorre un topo</i> | da 9) per mezzo della regola (11) |

linguaggio generato da una grammatica:

l'insieme di tutte e sole le stringhe che la grammatica consente di derivare

Una formulazione più sintetica delle regole di G :

$E \rightarrow SN SV$

$SN \rightarrow \text{Articolo Nome}$

$SV \rightarrow \text{VerbTrans SN} \mid \text{VerbIntr}$

$\text{VerbTrans} \rightarrow \text{rincorre} \mid \text{mangia}$

$\text{VerbIntr} \rightarrow \text{dorme} \mid \text{canta}$

$\text{Articolo} \rightarrow \text{il} \mid \text{un}$

$\text{Nome} \rightarrow \text{topo} \mid \text{gatto}$

G genera un **linguaggio finito**

Grammatica G'

alfabeto $S' = S \cup \{e, \textit{oppure}, \textit{ma}\}$

un nuovo simbolo non terminale: Connettivo

due nuove produzioni:

$$\begin{aligned} E &\rightarrow E \text{ Connettivo } E \\ \text{Connettivo} &\rightarrow e \mid \textit{oppure} \mid \textit{ma} \end{aligned}$$

Grammatica G''

alfabeto $S'' = S \cup \{che\}$

sostituiamo la seconda produzione di G con:

$$SN \rightarrow \text{Articolo Nome} \mid SN \text{ che } SV$$

NB. - la possibilità di generare infinite stringhe dipende dalla presenza di produzioni in cui il simbolo non terminale a sinistra della freccia compare anche nell'espressione che si trova sulla destra

Lo **stesso linguaggio** può essere generato da **grammatiche diverse**

Ad esempio: due grammatiche $G1$ e $G2$

Alfabeto: $\{a, b, c\}$

Simboli non terminali: S, A, B e C (S è l'assioma)

Produzioni:	$G1$	$G2$
	$S \rightarrow S1 C$	$S \rightarrow A S1$
	$S1 \rightarrow A B$	$S1 \rightarrow B C$
	$A \rightarrow aA \mid a$	$A \rightarrow aA \mid a$
	$B \rightarrow bB \mid b$	$B \rightarrow bB \mid b$
	$C \rightarrow cC \mid c$	$C \rightarrow cC \mid c$

Linguaggio generato: $aaabbbbbbcc, abbbbbcc, abccecc, abcc, \dots$

Si dicono **equivalenti** due grammatiche che generano lo stesso linguaggio

La gerarchia di Chomsky (1959)

grammatiche di tipo 0: nessun vincolo sulla forma delle produzioni

grammatiche di tipo 1 (grammatiche **non decrescenti**):

nessuna produzione $e_1 \rightarrow e_2$ in cui e_2 sia più corta di e_1

Ad es., $S a \rightarrow b$ non può far parte di una grammatica di tipo 1

G , G' e G'' sono grammatiche non decrescenti

Un altro esempio: grammatica GnD

alfabeto $\{a, b, c\}$

simboli non terminali: S, B (assioma: S)

produzioni:

$$S \rightarrow aSBc \mid abc$$

$$cB \rightarrow Bc$$

$$bB \rightarrow bb$$

linguaggio generato: $abc, aabbcc, aaabbbccc, \dots$,

Per ogni grammatica non decrescente, ne esiste una equivalente in cui tutte le produzioni hanno la forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

A: simbolo non terminale

α , β e γ : stringhe di simboli terminali e/o non terminali

Grammatiche di **tipo 1** = grammatiche **dipendenti dal contesto**

- o **sensibili al contesto** (*context sensitive*)

Grammatiche di tipo 2 (grammatiche libere dal contesto - *context free*)

Produzioni:

$$A \rightarrow \gamma$$

(caso particolare dello schema $\alpha A \beta \rightarrow \alpha \gamma \beta$ in cui $\alpha = \beta =$ stringa vuota)

Un **esempio**: grammatica *LC*

Alfabeto: $\{a, b\}$, unico simbolo non terminale l'assioma *S*

produzioni:

$$S \rightarrow aSb$$

$$S \rightarrow ab$$

Linguaggio generato: $ab, aabb, aaabbb, \dots$

Anche *G*, *G'* e *G''* sono libere dal contesto

Grammatiche di tipo 3 (grammatiche lineari)

- Grammatica *lineare destra* (o *regolare*): tutte le produzioni hanno la forma

$$A \rightarrow t B$$

oppure:

$$A \rightarrow t$$

- Grammatica *lineare sinistra*: tutte le produzioni hanno la forma

$$A \rightarrow B t$$

oppure:

$$A \rightarrow t$$

(A e B sono simboli non terminali, t simbolo terminale)

Per ogni grammatica lineare destra,
ne esiste una lineare sinistra equivalente, e viceversa

Un **esempio** di grammatica lineare (destra): grammatica *LD* che

Alfabeto $\{a, b\}$, simboli non terminali S e B (assioma: S)

Produzioni:

$$S \rightarrow aS$$

$$S \rightarrow B$$

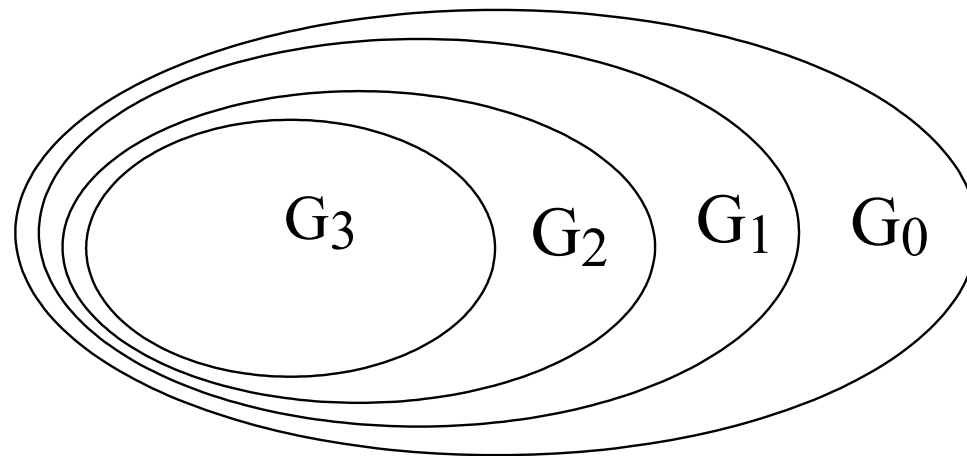
$$B \rightarrow bB$$

$$B \rightarrow b$$

Linguaggio generato: $ab, abb, aab, aabb, abbb, aaab, aabbb, \dots$

Vale che:

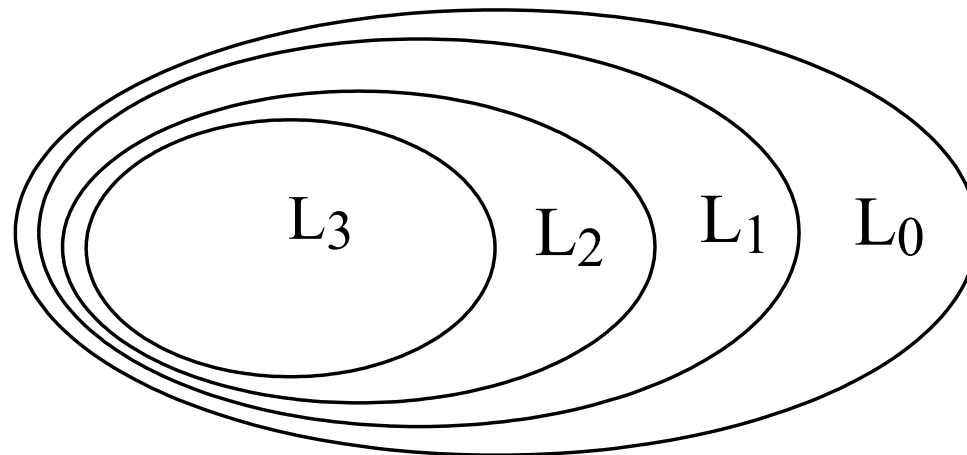
$$G_3 \subset G_2 \subset G_1 \subset G_0$$



Un **linguaggio è di tipo i** (con i che va da 0 a 3) se e solo se
esiste una grammatica di tipo i che lo genera

Vale che:

$$L_3 \subset L_2 \subset L_1 \subset L_0$$

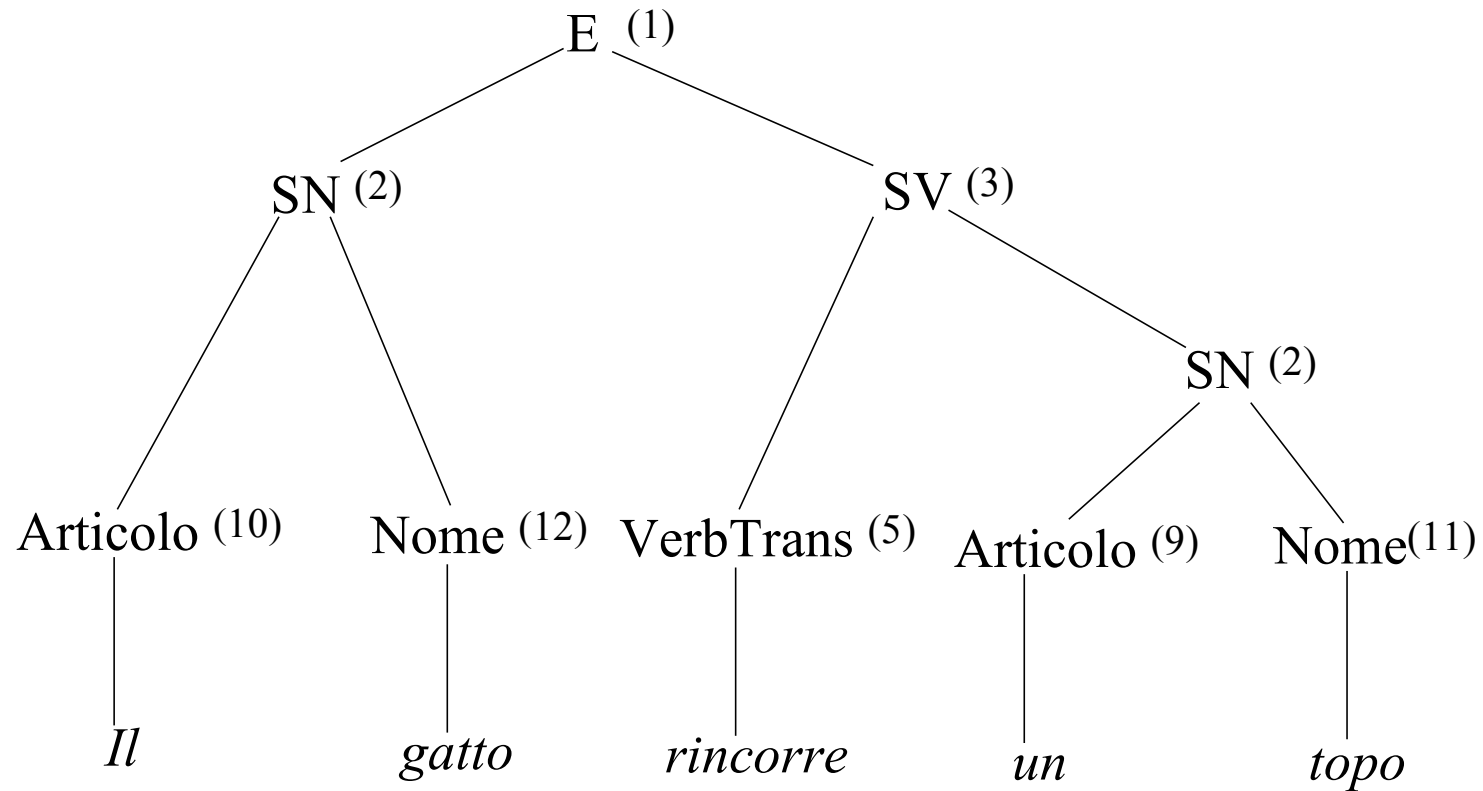


I linguaggi finiti sono un sottoinsieme proprio dei linguaggi di tipo 3

Alberi di derivazione

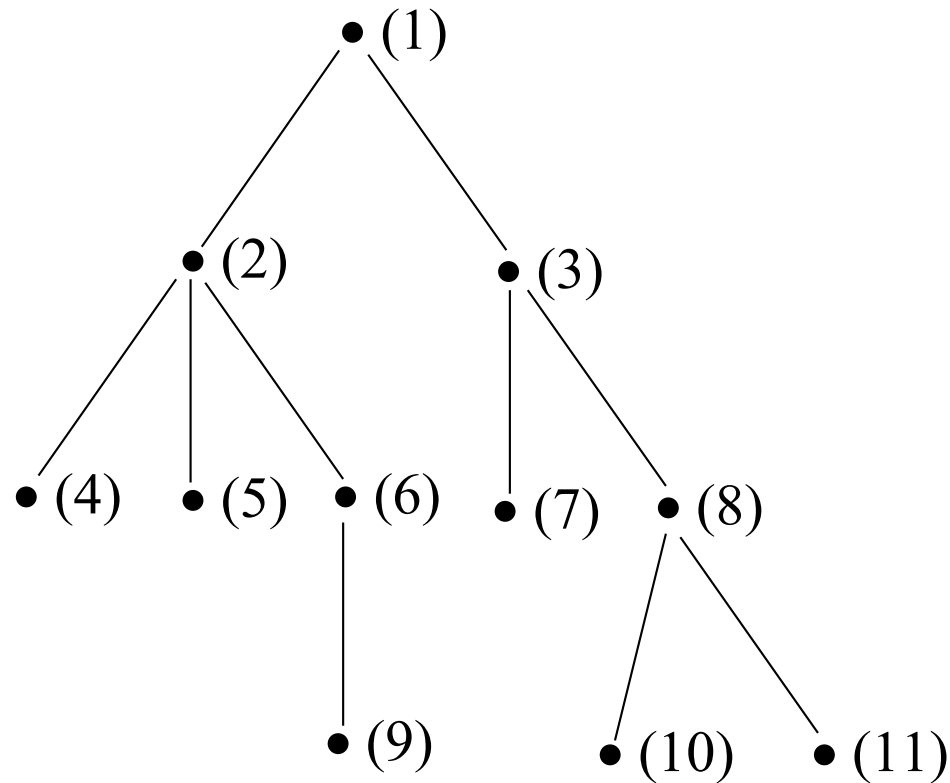
Nelle **grammatiche di tipo 2** (= libere dal contesto) la **derivazione** di una stringa può essere rappresentata per mezzo di un **albero** (**albero sintattico** o **albero di derivazione**)

Esempio: albero di derivazione in G della stringa *il gatto rincorre un topo*



- Foglie:** simboli terminali
- Altri nodi:** simboli non terminali
- Radice:** assioma

E' detto **albero** un grafo tale che i suoi **nodi** si possono disporre come nella figura



Il nodo (1) è detto **radice**. I nodi (2) e (3) sono i **successori** della radice; (4), (5) e (6) sono i successori di (2), eccetera. I nodi senza successori sono **foglie** (nella fig. le foglie sono (4), (5), (9), (7), (10) e (11))

Grammatiche ambigue:

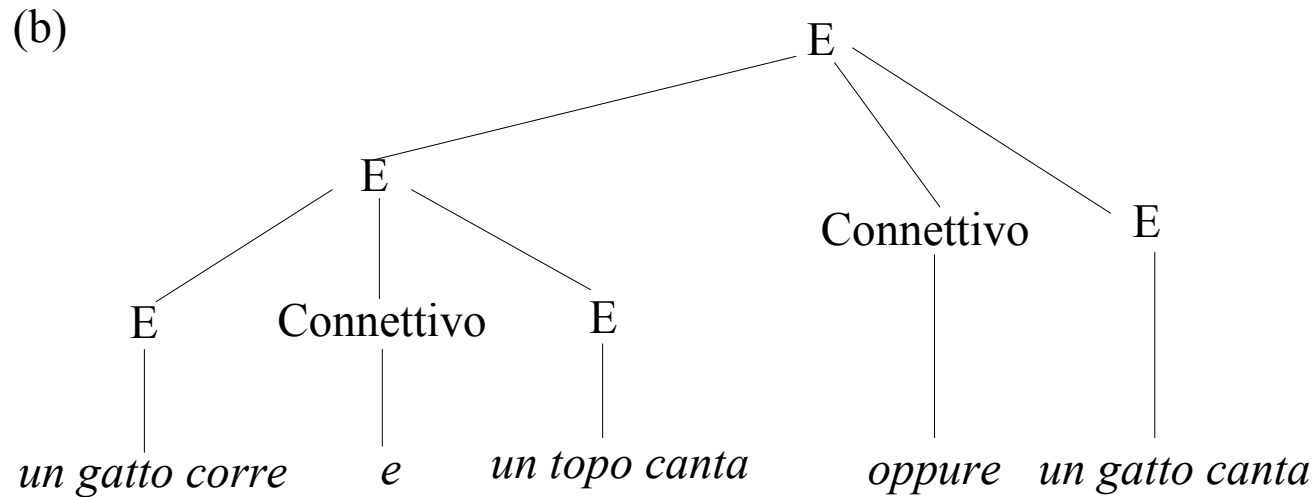
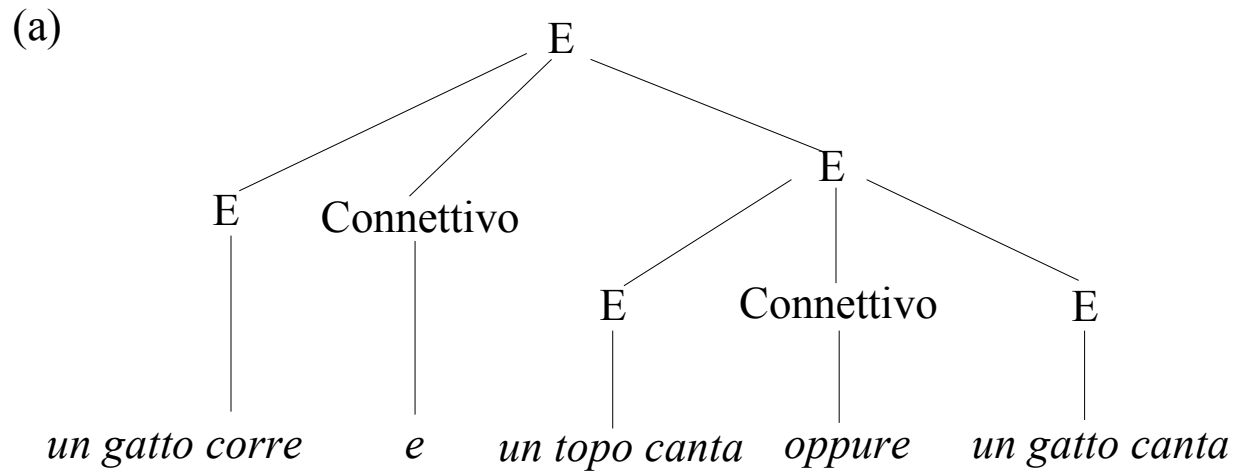
assegnano ad alcune stringhe strutture sintattiche diverse

Grammatiche ambigue libere dal contesto (= di tipo 2):

associano ad alcune stringhe alberi di derivazione
differenti

Un **esempio** di grammatica ambigua: G'

un gatto corre e un topo canta oppure un gatto canta



Grammatiche e macchine

Due tipi di problemi:

- **Generare un linguaggio**
- **Riconoscere (o accettare) un linguaggio**

Riconoscere un linguaggio è più difficile che generarlo

Insiemi ricorsivamente numerabili e insiemi ricorsivi

- insieme **ricorsivamente enumerabile**:

esiste una MT che ne genera tutti gli elementi

- insieme **ricorsivo**:

esiste una MT che, dato un oggetto, decide se appartiene all'insieme o meno

(Tutti gli insiemi ricorsivi sono ricorsivamente enumerabili, ma non viceversa)

Quindi:

- un **linguaggio** è **generato da una MT**
se e solo se è un insieme **ricorsivamente enumerabile**
- un **linguaggio** è **accettato da una MT** se e solo se è un insieme **ricorsivo**

Il **linguaggio generato da una grammatica di tipo 0** costituisce un insieme **ricorsivamente enumerabile** (= può essere generato da una MT)

Traccia di una **dimostrazione**:

data una qualsiasi grammatica a struttura di frase, si può definire un algoritmo che generi il linguaggio corrispondente:

- si parte dall'assioma;
- si selezionano tutte le produzioni che possono essere applicate all'assioma, e si applicano una dopo l'altra ottenendo un insieme (finito) E_1 di espressioni;
- per ciascun elemento di E_1 si selezionano tutte le produzioni che possono essergli applicate; poi si applicano una dopo l'altra agli elementi di E_1 le produzioni corrispondenti ottenendo un nuovo insieme E_2 (sempre finito) di espressioni,

e così via

Man mano che si ottiene una stringa di soli simboli terminali la si produce in output

In base alla Tesi di Church, tutto ciò che può essere fatto da un algoritmo può essere fatto anche da una MT, per cui ogni linguaggio generato da una grammatica di tipo 0 può essere generato anche da una MT

(è possibile una dimostrazione diretta, che non faccia appello alla Tesi di Church)

Il linguaggio generato da una grammatica di tipo 1 (non decrescente)

costituisce un insieme ricorsivo (= può essere accettato da una MT)

Traccia di una **dimostrazione**:

Data una grammatica non decrescente, si può definire un algoritmo che accetti il linguaggio corrispondente:

Preso in input una stringa s di lunghezza n la sua lunghezza, si applica un procedimento simile a quello dell'algoritmo precedente. Quando si ottiene un'espressione la cui lunghezza è maggiore di n , la si scarta

- Se s è una stringa del linguaggio, verrà generata in un numero finito di passi
- Se s non fa parte del linguaggio, verrà un momento in cui non verranno più prodotte stringhe nuove poiché tutte le espressioni ottenute superano la lunghezza n

In base alla Tesi di Church, tutto ciò che può essere fatto da un algoritmo può essere fatto anche da una MT, per cui ogni linguaggio generato da una grammatica di tipo 0 può essere accettato da una MT

(è possibile una dimostrazione diretta, che non faccia appello alla Tesi di Church)

- Per **riconoscere** (o accettare) un **linguaggio di tipo 2**
è sufficiente un **automa a pila**

- Per **riconoscere** (o accettare) un **linguaggio di tipo 3**
è sufficiente un **automa a stati finiti**

Automati a stati finiti (o automi finiti, AF)

(consideriamo AF che *riconoscono*, o *accettano*, le stringhe di un linguaggio)

stati interni: $Q = \{q_0, \dots, q_n\}$

stato iniziale: q_0

stati finali: ...

istruzioni: $q_i \quad s_j \quad q_k$

Quando un automa AF_L si ferma, se la stringa S degli input è stata esaminata interamente, e se AF_L si trova in uno stato finale, allora S fa parte del linguaggio L riconosciuto (o accettato) da AF_L , altrimenti S non fa parte di L

Un **esempio**: automa AF_{LD} che accetta il linguaggio generato dalla grammatica LD

tre stati: q_0, q_1, q_2

(unico) stato finale: q_2

istruzioni:

- 1) $q_0 \ a \ q_1$
- 2) $q_1 \ a \ q_1$
- 3) $q_1 \ b \ q_2$
- 4) $q_2 \ b \ q_2$

input: stringa $S = aabbb$

$a \ a \ b \ b \ b$
↑
 q_0

$a \ a \ b \ b \ b$
↑
 q_1

$a \ a \ b \ b \ b$
↑
 q_1

$a \ a \ b \ b \ b$
↑
 q_2

.....

Stringhe come $abba$, $aaaa$ e $bbbaa$ non vengono accettate da AF_{LD}

Un AF può essere rappresentato con un **diagramma a stati**

stati non terminali: cerchi semplici

stati terminali: cerchi doppi

istruzioni $q_i \ s_j \ q_k$: un arco orientato contrassegnato con s_j che va da q_i a q_k

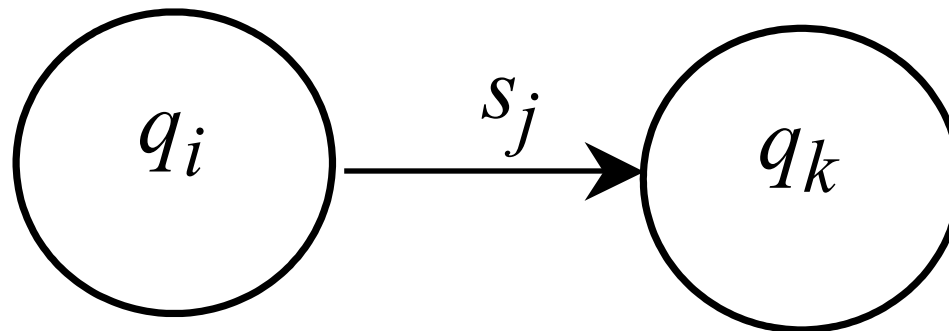
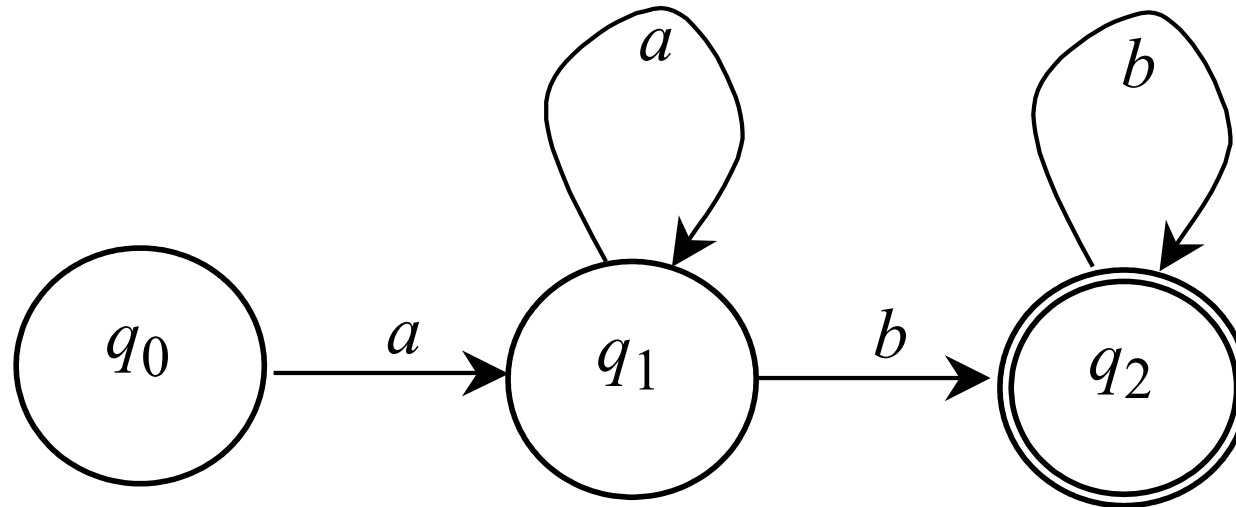


Diagramma di AF_{LD}



una **stringa** s_1, \dots, s_k è **accettata** da un AF se e solo se nel corrispondente diagramma esiste un percorso dallo stato iniziale a uno stato finale, in cui gli archi attraversati sono contrassegnati, nell'ordine, con i simboli s_1, \dots, s_k

Automati a pila AP (*pushdown automata*)

stati interni: $Q = \{q_0, \dots, q_n\}$

stato iniziale: q_0

stati finali: \dots

alfabeto della pila: $Al_P = \{A_1, \dots, A_m\}$

istruzioni: $(q_i, s_j, A_k) \rightarrow (q_l, A_h \dots A_t)$

Sono possibili istruzioni del tipo:

$$(q_i, s_j, A_k) \rightarrow (q_l, -)$$

e

$$(q_i, s_j, -) \rightarrow (q_l, A_h \dots A_t)$$

All'inizio del calcolo la **pila** deve essere **vuota**

Una **stringa fa parte del linguaggio accettato da un automa AP** se e solo se, quando AP si ferma, sono vere contemporaneamente queste **tre condizioni**:

- a) tutta la stringa di input è stata letta
- b) l'automata è in uno stato finale
- c) la pila è vuota

Esempio: automa AP_{LC} che accetta il linguaggio LC

due **stati interni**; q_0 e q_1

unico **stato finale**: q_1

A è l'unico **simbolo** che AP_{LC} può scrivere nella sua **pila**

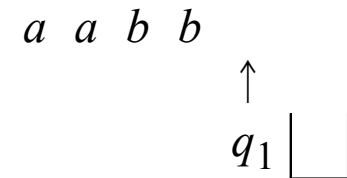
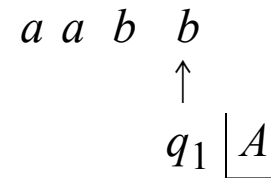
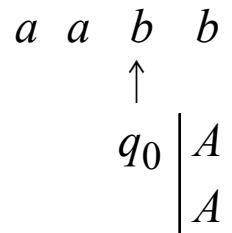
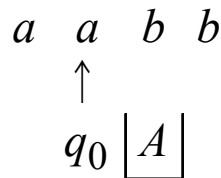
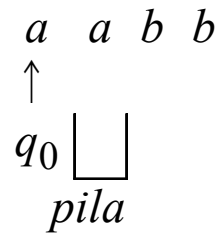
istruzioni:

$$1) (q_0, a, -) \rightarrow (q_0, A)$$

$$2) (q_0, b, A) \rightarrow (q_1, -)$$

$$3) (q_1, b, A) \rightarrow (q_1, -)$$

Calcolo di AP_{LC} con input $aabb$:



NB: LC non può essere accettato da alcun AF

MT

più potenti degli

AP

più potenti degli

AF

Ricapitolando:

<i>Tipo di linguaggio</i>	
Tipo 3 (Lineare)	Accettato da un automa a stati finiti
Tipo 2 (Libero dal contesto)	Accettato da un automa a pila
Tipo 1 (Dipendente dal contesto)	Accettato da una MT (<u>tutti</u> i linguaggi di tipo 1 sono ricorsivi)
Tipo 0	Generato da una MT (i linguaggi di tipo 0 sono ricorsivamente enumerabili ma, in generale, non ricorsivi)