Grammatiche

- Gli automi sono un modello riconoscitivo/traduttivo/*elaborativo* (di linguaggi):
  essi "ricevono" una stringa nel loro ingresso e la elaborano in vari modi
- Passiamo ora ad esaminare un modello *generativo*:
  una grammatica produce o *genera* stringhe (di un linguaggio)
- Concetto generale di ***grammatica*** o ***sintassi*** (matematicamente, alfabeto e vocabolario e grammatica e sintassi sono sinonimi):
  insieme di regole per costruire frasi di un linguaggio (stringhe): si applica a qualsiasi nozione di linguaggio nel senso più lato.
- In modo sostanzialmente simile ai normali meccanismi linguistici una grammatica formale genera stringhe di un linguaggio attraverso un processo di *riscrittura*:

- “Una frase è costituita da un soggetto seguito da un predicato
  Un soggetto a sua volta può essere un sostantivo, oppure un pronome, oppure ….
  Un predicato può essere costituito da un verbo seguito da un complemento …”
- Un programma è costituito da una parte dichiarativa e da una parte esecutiva
  La parte dichiarativa …
  La parte esecutiva è costituita da una sequenza di istruzioni
  Un’istruzione può essere semplice o composta ….

- Un messaggio di posta elettronica è costituito da una testata e da un corpo
  La testata contiene indirizzo, ….
- In generale questo tipo di regole linguistiche descrive un "oggetto principale" (libro, programma, messaggio, protocollo, ...) come una sequenza di "oggetti componenti" (soggetti, testate, parti dichiarative, …). Ognuno di questi viene poi "raffinato" rimpiazzandoli con altri oggetti più dettagliati e così via finché non si giunge ad una sequenza di elementi base (bit, caratteri, …)
  Le varie riscritture possono presentare delle alternative: un soggetto può essere un nome o un pronome o altro, un'istruzione può essere di assegnamento o di I/O, ...

La definizione formale di grammatica

- $G = (V_N, V_T, P, S)$
  - $\mathbf{V_N}$: alfabeto o vocabolario *nonterminale*
  - $\mathbf{V_T}$: alfabeto o vocabolario *terminale*
  - $V = V_N \cup V_T$
  - $\mathbf{S \in V_N}$: elemento particolare di V_N detto *assioma o simbolo iniziale*
  - $\mathbf{P \subseteq V_N^+ \times V^*}$: insieme delle regole di riscrittura, o *produzioni*
  - Per comodità scriveremo $\boldsymbol{\alpha \rightarrow \beta}$ al posto di (α, β)

Esempio

- $V_N = \{S, A, B, C\}$
- $V_T = \{a,b,c,d\}$
- S
- $P = \{S \rightarrow AB,\ BA \rightarrow cCd,\ CBS \rightarrow ab,\ A \rightarrow \varepsilon\}$

Relazione di derivazione immediata

$\alpha \Rightarrow \beta$, $\alpha \in V^+$, $\beta \in V^*$

se e solo se

$\alpha = \alpha_1\ \alpha_2\ \alpha_3$, $\beta = \alpha_1\ \beta_2\ \alpha_3$, $\alpha_2 \rightarrow \beta_2 \in P$

α_2 si riscrive come β_2 nel contesto (α_1, α_3)

Rispetto alla grammatica precedente:

aaBAS $\Rightarrow$ aacCdS

Definiamo poi come al solito la chiusura riflessiva e transitiva di $\Rightarrow$: $\Rightarrow^*$

Linguaggio generato da una grammatica

$$L(G) = \{x \in V_T^* \mid S \Rightarrow^* x\}$$

Consiste di tutte le stringhe costituite da soli simboli terminali derivabili (in numero qualsiasi di passi) da S

Primo esempio

$G_1 = (\{S, A, B\}, \{a, b, 0\},$
$\{S \rightarrow aA, A \rightarrow aS, S \rightarrow bB, B \rightarrow bS, S \rightarrow 0\}, S)$

Alcune derivazioni

$S \Rightarrow 0$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aa0$

$S \Rightarrow bB \Rightarrow bbS \Rightarrow bb0$

$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabb0$

Mediante una facile generalizzazione:

$L(G_1) = \{aa, bb\}^* \cdot 0$

Secondo esempio

$G_2 = (\{S\}, \{a, b\},$
$\{S \rightarrow aSb \mid ab\}, S)$ (abbreviazione per $S \rightarrow aSb$, $S \rightarrow ab$)

Alcune derivazioni
$S \Rightarrow ab$
$S \Rightarrow aSb \Rightarrow aabb$
$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$

Mediante una facile generalizzazione:
$L(G_2) = \{a^n b^n \mid n > 0\}$
Se sostituiamo $S \rightarrow ab$ con $S \rightarrow \varepsilon$ otteniamo:
$L(G_2) = \{a^n b^n \mid n \geq 0\}$

Terzo esempio

$S \rightarrow aACD$, $A \rightarrow aAC$, $A \rightarrow \varepsilon$,
$CD \rightarrow BDc$, $CB \rightarrow BC$, $B \rightarrow b$, $D \rightarrow \varepsilon$

Alcune derivazioni
$S \Rightarrow aACD \Rightarrow aCD \Rightarrow aBDc \Rightarrow abDc \Rightarrow abc$
$S \Rightarrow aACD \Rightarrow aCD \Rightarrow aC$ no
$S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCCD \Rightarrow aaCBDc \Rightarrow aaCbDc$ no
$S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCCD \Rightarrow aaCBDc \Rightarrow aaBCDc \Rightarrow$
$aaBBDcc \Rightarrow aaBBcc \Rightarrow^2 aabbcc$

Di che linguaggio si tratta?

Alcune domande “naturali”

- Quale utilità pratica delle grammatiche (oltre ai “divertimenti” con $\{a^n b^n\}$?)
- Quali linguaggi si possono ottenere con le grammatiche?
- Che relazioni esistono tra grammatiche e automi (o meglio tra i linguaggi *generati* dalle grammatiche e i linguaggi *riconosciuti* dagli automi)?

Alcune risposte

- Definizione della sintassi dei linguaggi di programmazione
- Applicazioni duali rispetto a quelle degli automi
- Esempio più semplice: la *compilazione* dei linguaggi (non solo di programmazione): la grammatica definisce il linguaggio; l'automa lo riconosce e traduce.
- In modo un po' più sistematico:

Classi di grammatiche

- **Grammatiche non-contestuali** (*context-free*):
  - Ogni produzione ha la forma $\alpha \rightarrow \beta$ dove $|\alpha| = 1$, cioè α è un elemento di V_N.
  - Non-contestuale perché la riscrittura di α non dipende dal contesto in cui si trova.
  - Sono di fatto la stessa cosa della BNF usata per definire la sintassi dei linguaggi di programmazione (quindi vanno bene per definire certi meccanismi sintattici tipici dei linguaggi di programmazione e naturali … ma non tutti)
  - Le G_1 e G_2 precedenti sono non-contestuali, non così la G_3.

Grammatiche regolari:

- Ogni produzione ha la forma $\alpha \rightarrow \beta$, dove
  $|\alpha| = 1$, $\beta \in V_T . V_N \cup V_T$.

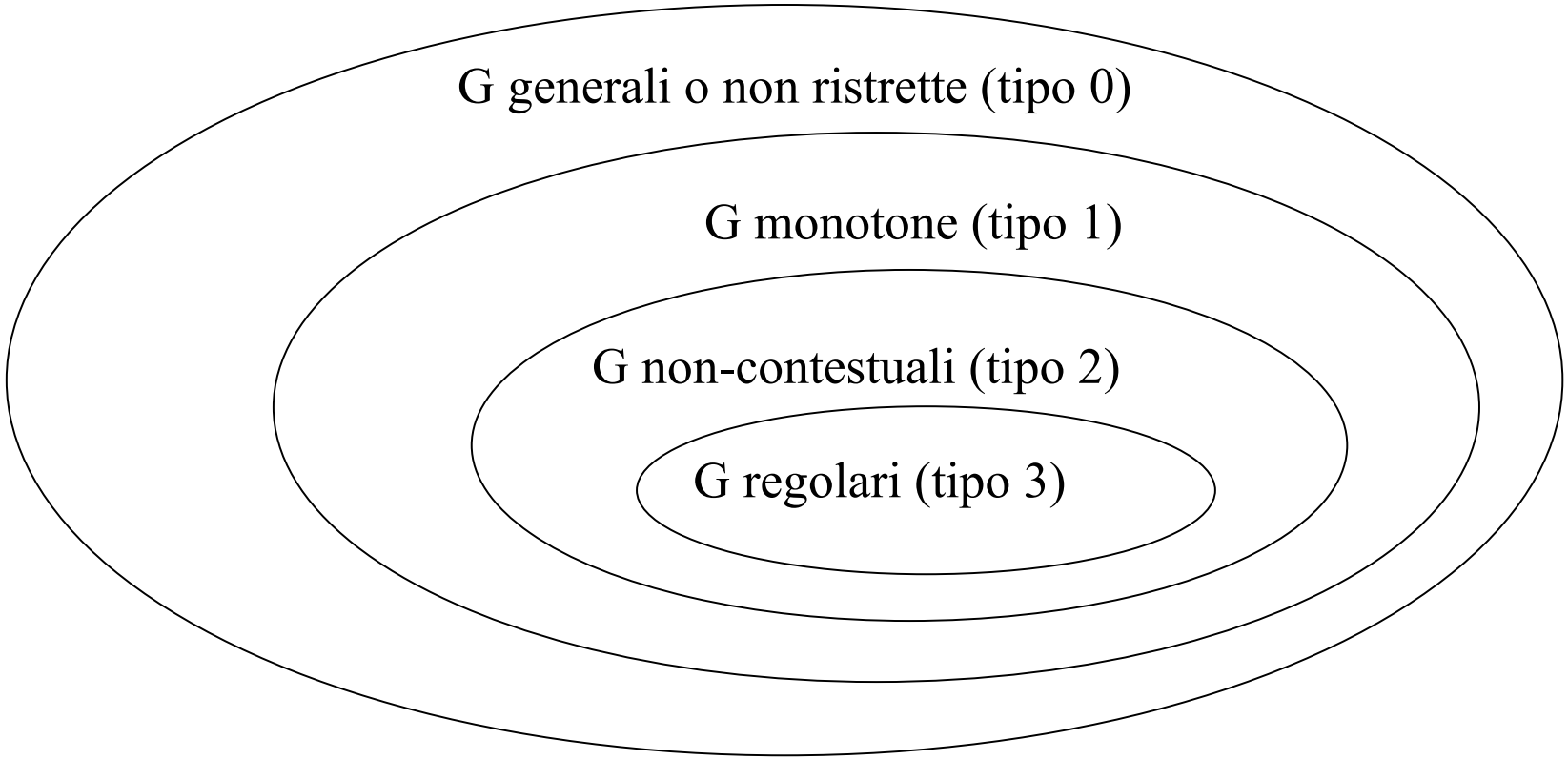
- Le grammatiche regolari sono anche non-contestuali, ma non viceversa
- La G_1 precedente è regolare, ma non la G_2.
- Per la stringa vuota si deve ammettere anche $S \rightarrow \varepsilon$

Grammatiche monotone:

- Ogni produzione ha la forma $\alpha \rightarrow \beta$, dove $|\alpha| \leq |\beta|$
- Per la stringa vuota si deve ammettere $S \rightarrow \varepsilon$, però S non deve apparire in parti destre di produzioni

- Le *grammatiche regolari* sono chiaramente monotone
- quelle *non-contestuali* non lo sono, perché si possono avere produzioni del tipo $A \rightarrow \varepsilon$. Queste però si possono eliminare abbastanza facilmente.
  - es: se ho una produzione B → abaAcDA, la rimpiazzo con le produzioni B → abaAcDA | abacDA | abaAcD | abacD
- Qualche minimo cenno, poi non ne parliamo più:
  - i linguaggi delle g. monotone sono anche detti *contestuali* e sono definiti da *Automi Lineari* (MT con lunghezza del nastro *limitata*)

Relazioni tra grammatiche e linguaggi (gerarchia di Chomsky)



Se ne deduce immediatamente che:

$$L_3 \subseteq L_2 \subseteq L_1 \subseteq L_0$$

Ma i contenimenti sono stretti?

La risposta dal confronto con gli automi

Relazioni tra grammatiche e automi (senza troppe sorprese)

- G. Regolari (GR) equivalenti agli Automi a Stati Finiti
  - Dato un ASF A, poniamo $V_N = Q$, $V_T = I$, $S = q_0$, e,
    per ogni $\delta(q, i) = q'$ poniamo
    $q \rightarrow i\ q'$
    inoltre se $q' \in F$, aggiungiamo $q \rightarrow i$
  - è intuitivo (facile induzione) che
    $\delta^*(q, x) = q'$ se e solo se $q \Rightarrow^* x\ q'$
  - Viceversa:
  - Data una GR, poniamo $Q = V_N \cup \{q_F\}$, $I = V_T$, $q_0 = S$, $F = \{q_F\}$ e,
    per ogni $A \rightarrow bC$ poniamo $\delta(A, b) = C$
    per ogni $A \rightarrow b$ poniamo $\delta(A, b) = q_F$

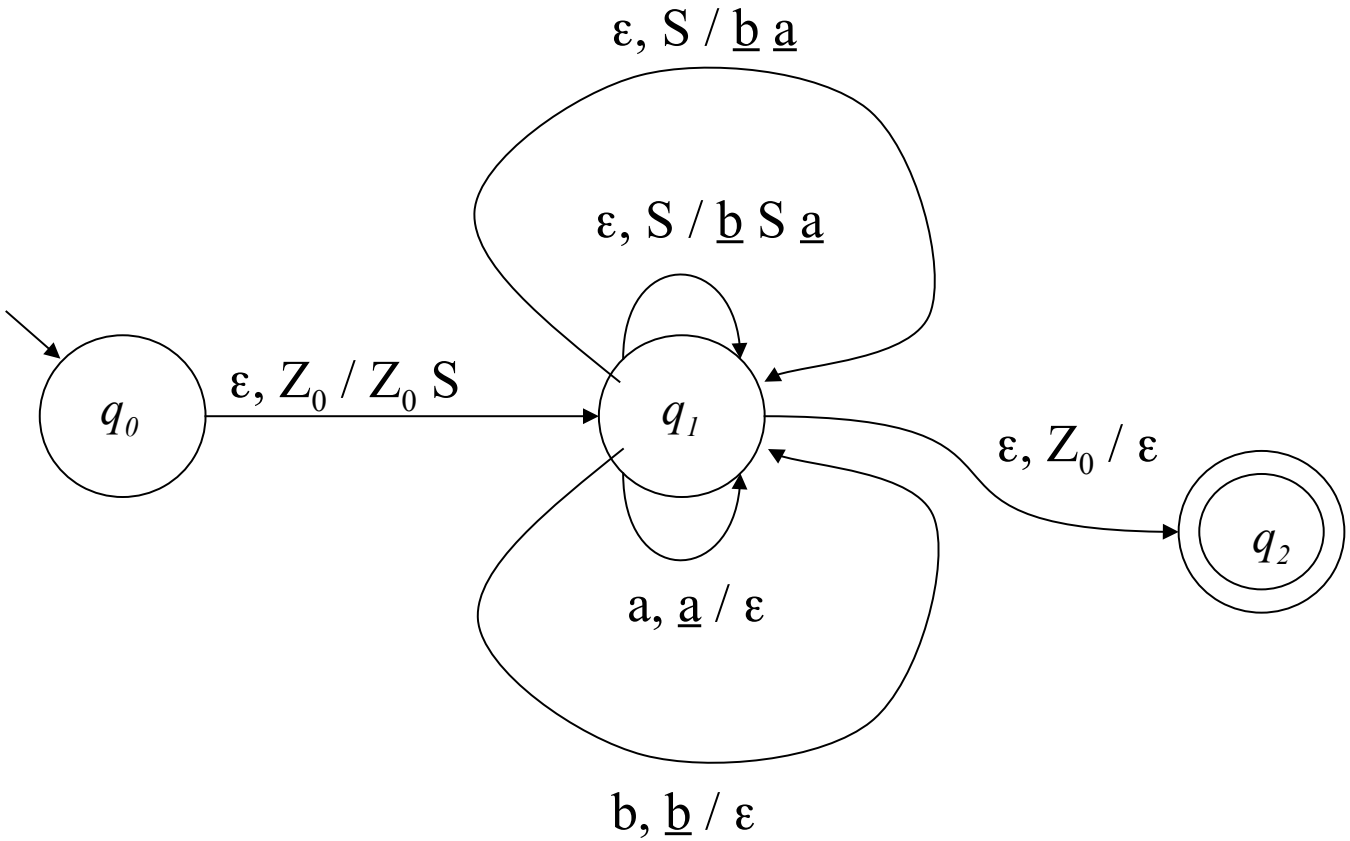
NB: l'ASF così ottenuto è nondeterministico: molto più comodo!

- GNC equivalenti a AP (ND!)

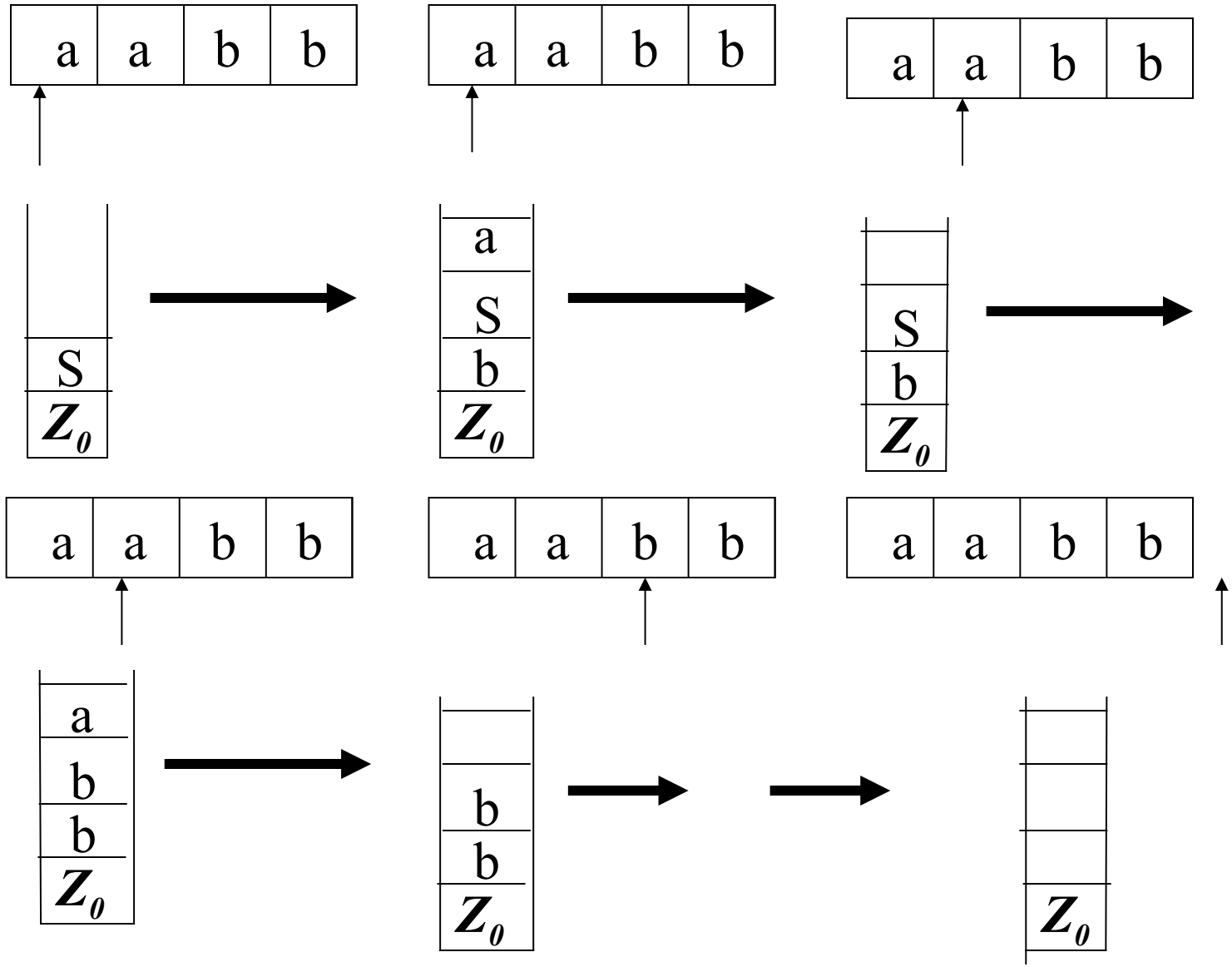
giustificazione intuitiva (senza dimostrazione:
la dimostrazione costituisce il “cuore” della costruzione di un compilatore):

consideriamo la grammatica:

$S \rightarrow a\ S\ b \mid ab$


ε, S / b a
ε, S / b S a
q0
ε, Z0 / Z0 S
q1
ε, Z0 / ε
q2
a, a / ε
b, b / ε

$$S \Rightarrow aSb \Rightarrow aabb$$



Ggen equivalenti alle MT

- Data G costruiamo (a grandi linee) una MT a nastro singolo *non deterministica*, M, che accetti L(G):

  La stringa x si trova nel nastro alla partenza.

  – Ciclo:
  – Il nastro (durante il funzionamento esso conterrà una stringa $\gamma \in V^*$) viene scandito alla ricerca di una parte *destra* β di qualche produzione $\alpha \rightarrow \beta$ di P
  – Quando se ne trova una -*non necessariamente la prima, operando una scelta ND*- essa viene sostituita dalla corrispondente parte sinistra α

- In tal modo: $\gamma \Rightarrow \delta$ sse $c = \langle q, \delta \rangle \vdash^* \langle q, \gamma \rangle$
- in pratica seguo la relazione di derivazione $\Rightarrow$ *da destra verso sinistra*

Se e quando il contenuto del nastro diviene l'assioma S, x viene accettata, cioè $\langle q_0, x \rangle \vdash^* \langle q_F, S \rangle$, altrimenti questa particolare sequenza di mosse non porta all'accettazione.

Nota: in caso di grammatiche *monotone*, useremo solo la memoria occupata inizialmente da x, non celle ulteriori (Automa Lineare), a parte il caso particolare della stringa vuota.

• è essenziale -e, vedremo, ineluttabile- il fatto che, se $x \notin L(G)$, M potrebbe "tentare infinite strade", alcune delle quali anche non terminanti, senza poter concludere che $x \in L(G)$ (giustamente), ma *neanche il contrario*.

Infatti la definizione di accettazione richiede che M giunga in una configurazione di accettazione se e solo se $x \in L$, ma non richiede che M termini la computazione (in uno stato negativo) se $x \notin L$

• Tornano in ballo il problema-complemento e l'asimmetria tra risolvere un problema in maniera positiva o in maniera negativa.

- Viceversa: data MT M (per comodità e senza perdere generalità, a nastro singolo) costruiamo (a grandi linee) una G, che generi L(M):

- In primo luogo G genera *tutte* le stringhe del tipo x$X, $x \in V_T^*$

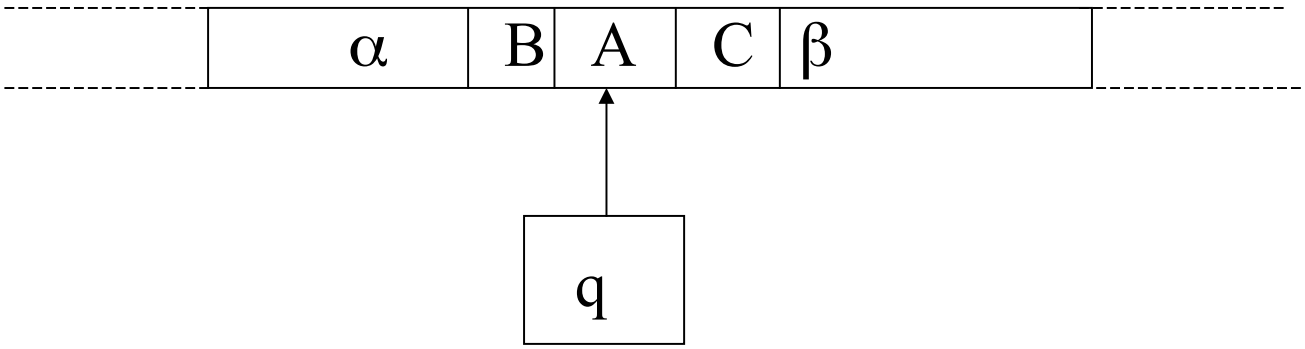
  X essendo una “copia di x” costituita da caratteri nonterminali (ad esempio, per x = aba, x$X = aba$ABA).
- In effetti G lavora su X per simulare M, potendo "manipolare" solo nonterminali.

(inciso: es. di grammatica x$X, con alfabelto terminale {a,b})

$S \rightarrow S\ A'\ A$	$S \rightarrow S\ B'\ B$
$S \rightarrow \$$	
$A\ A' \rightarrow A'\ A$	$A\ B' \rightarrow B'\ A$
$B\ A' \rightarrow A'\ B$	$B\ B' \rightarrow B'\ B$
$\$A' \rightarrow a\$$	$\$B' \rightarrow b\$$

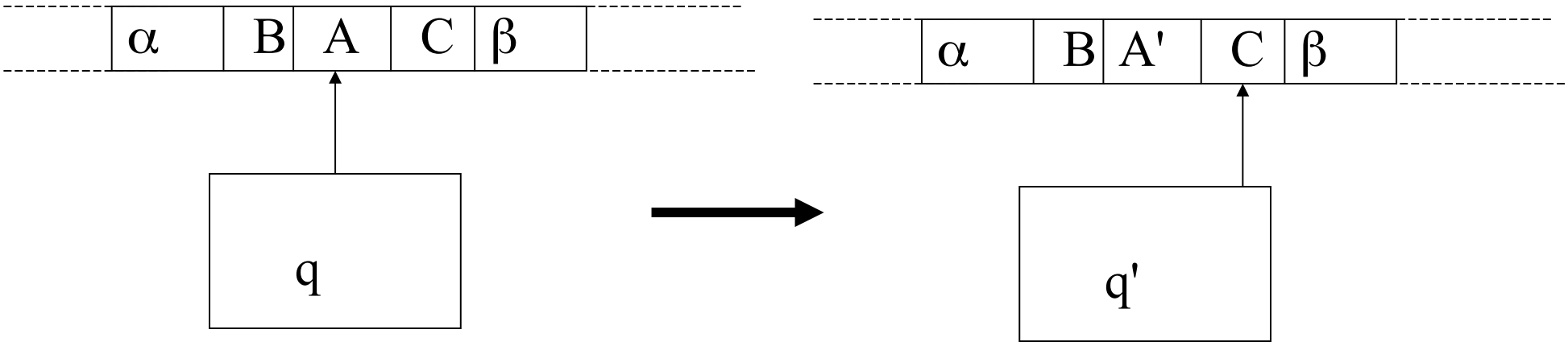
- L'obiettivo è ricavare da x$X una derivazione $x\$X \Rightarrow^* x$ se e solo se x è accettata da M.
- L'idea base è di simulare ogni mossa di M mediante una derivazione immediata di G:

- Rappresento la configurazione



- Mediante la stringa (i casi particolari sono lasciati in esercizio): $αBqACβ
- Se è definita:
  - δ(q, A) = <q', A', R> si definisce in G la produzione qA → A' q'
  - δ(q, A) = <q', A', S> si definisce in G la produzione qA → q' A'
  - δ(q, A) = <q', A', L> si definisce in G la produzione BqA → q' BA'
    ∀ B dell'alfabeto di M (si ricordi che M è a nastro singolo e quindi ha un solo alfabeto per ingresso, memoria e uscita)

– In tal modo:



– Se e solo se: x\$αBqACβ ⇒ x\$αBA'q'Cβ, ecc.

– Aggiungo infine produzioni che permettano a G di derivare da x\$αBq$_F$ACβ la sola x se -e solo se- M giunge a una configurazione di accettazione (αBq$_F$ACβ), cancellando tutto ciò che si trova a destra di \$, \$ compreso.