Calcolabilità e Linguaggi Formali

Giacomo De Liberali

23 ottobre 2017

Indice

1 Introduzione

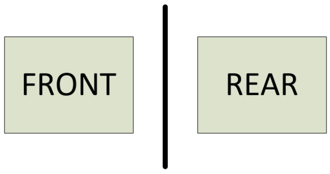
21 Settembre 2017

Gli argomenti trattati in questo corso sono:

1. Teoria degli automi (capitoli $1, 2, 3$)
2. Calcolabilità: cosa si può computare? (capitili $3, 4, 5$)
3. Complessità delle soluzioni

2 Linguaggi regolari

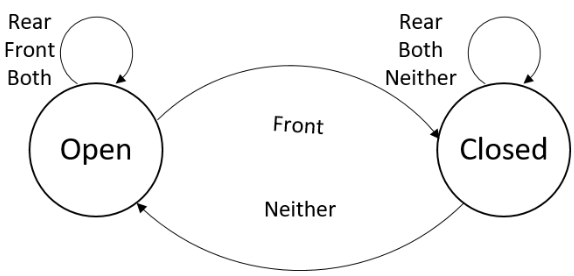
Il modello di computazione più elementare sono gli automi a stati finiti. Andiamo a rappresentare tramite un automa una porta con una bilancia da ogni lato che permetta l'accesso alle persone e impedisca alla porta stessa di colpire qualcuno:



La pressione della pedana *Font* comporterà quindi l'apertura della porta. I possibili input sono quattro:

1. Front
2. Rear
3. Both
4. Neither

e rappresentati in un automa:



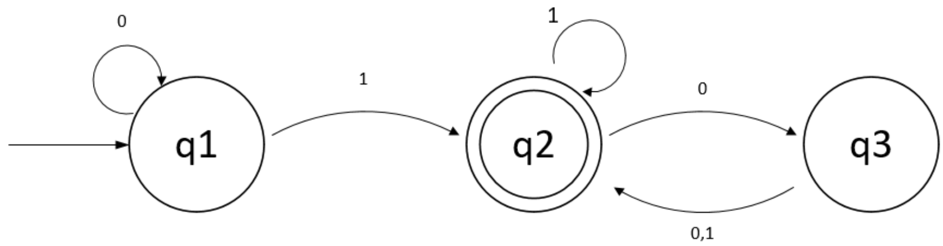
3 Automa a stati finiti

3.1 Definizione

Un automa a stati finiti è una 5-tupla $(Q,\ \Sigma,\ \delta,\ q_0,\ F)$ dove:

1. Q è un insieme finito di stati
2. Σ è un insieme finito di simboli (detto *alfabeto*)
3. δ è la *funzione di transizione* che data una coppia stato-alfabeto ritorna uno stato: $\delta : Q \times \Sigma \to Q$
4. $q_0 \in Q$ è lo stato iniziale (negli automi è ammesso solo uno stato iniziale)
5. $F \subseteq Q$ è un insieme degli stati finali (o *accettanti*)

Esempio.



L'automa può essere rappresentato formalmente mediante la seguente sintassi:

$$A = (\{q_1, q_2, q_3\},\ \{0,1\},\ \delta,\ q_1,\ \{q_2\})$$

dove δ è definita come:

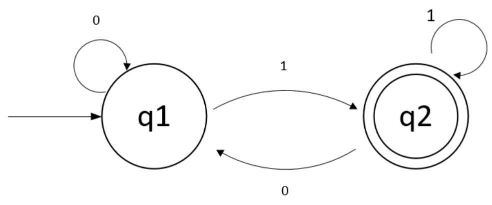
$$\begin{array}{ll} \delta(q_1,0) = q_1 & \delta(q_1,1) = q_2 \\ \delta(q_2,0) = q_3 & \delta(q_2,1) = q_2 \\ \delta(q_3,0) = q_2 & \delta(q_3,1) = q_2 \end{array}$$

Analizzando le stringhe che l'automa accetta, possiamo notare che quelle valide (e quindi accettate) sono tutte quelle che

1. finiscono con 1
2. oppure finiscono con 0 e hanno un numero pari di 0 dopo l'ultimo 1

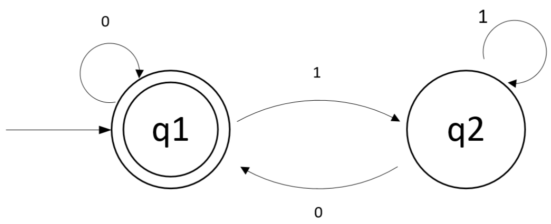
Ad esempio, la stringa 1101 è una stringa valida in quanto alla fine dell'input l'automa è nello stato q_2 che è l'unico stato finale.

Esempio.



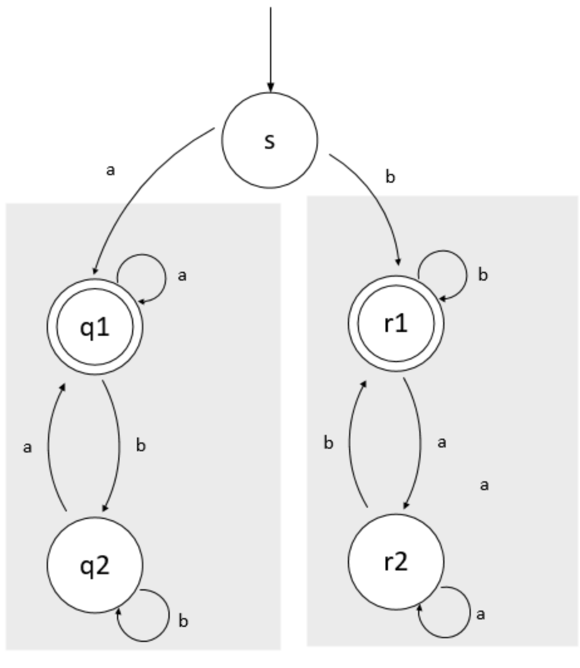
Questo automa riconosce stringhe in $\{0, 1\}$ che finiscono per 1. Nel caso di stringa vuota (ε) non entro in nessuno stato e non mi muovo.

Esempio.



Questo automa riconosce stringhe in $\{0, 1\}$ che finiscono per 0 oppure ε.

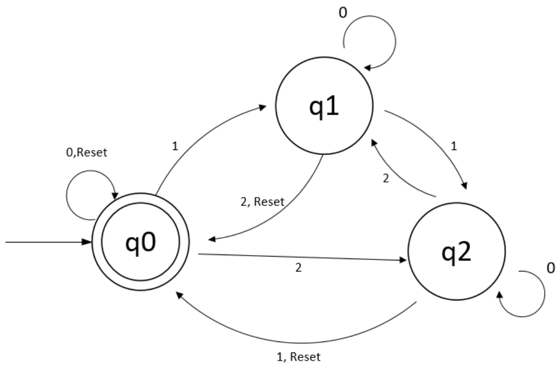
Esempio.



Questo automa riconosce stringhe in $\{a, b\}$ che iniziano e finiscono per lo stesso simbolo. In automi come in questo caso è utile poter scomporre il problema in più sotto-problemi Possiamo notare infatti che una volta scelto uno dei due rami che non è possibile passare all'altro. Questo suggerisce di studiare singolarmente i due automi sinistro e destro e infine di comporre la soluzione.

L'automa di sinistra accetta stringhe che iniziano e finiscono solo per a, mentre l'automa di sinistra solo stringhe che iniziano e finiscono per b.

Esempio.



Questo automa fa la somma modulo 3 di tutti i numeri letti in input dopo l'ultimo simbolo di *Reset*, se esiste.

3.2 Accettazione di una stringa

Sia $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ un automa a stati finiti e sia $w = w_1, ..., w_n$ una stringa tale che

$$\forall i \in [1...n] : w_i \in \Sigma$$

Diciamo che M accetta w se e solo se esiste una sequenza di stati $R_0, R_1, ..., R_n \in Q$ tali che

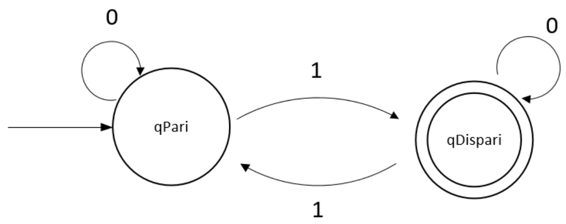
1. $R_0 = q_0$ (partendo dal nodo iniziale)
2. $R_n \in F$ (terminando in uno stato finale)
3. $\forall i \in [0, n-1] : \sigma(R_i, w_{i+1}) = R_{i+1}$ (per ogni input la *funzione di transizione* termini in uno stato finale)

M riconosce il linguaggio A se e solo se

$$A = \{w \mid M \text{ accetta } w\}$$

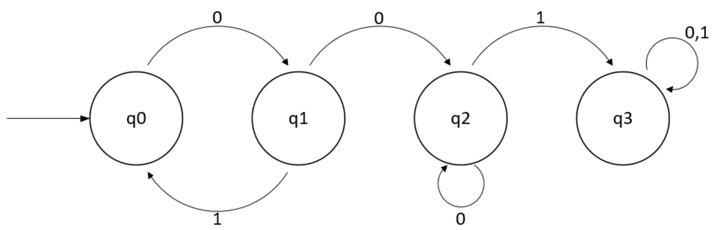
Un linguaggio A è *regolare* se e solo se esiste un automa a stati finiti M tale che M riconosce A.

Esempio. : alfabeto $\{0, 1\}$ e vogliamo riconoscere tutte le stringhe con un numero dispari di 1. L'idea è di avere un bit di informazione, quindi due stati, che mi rappresentano la condizione: ho contanto un numero pari di 1 oppure ho contanto un numero dispari di 1. L'automa è il seguente:



Esempio. : alfabeto $\{0, 1\}$ e vogliamo riconoscere tutte le stringhe che contengono almeno una volta la sotto-stringa 001. L'invariante è:

1. q_0: non ho letto sequenze di 001
2. q_1: ho letto 0
3. q_2: ho letto 00
4. q_3: ho letto 001



3.3 Operazioni regolari

Siano A e B due linguaggi. Definiamo le operazioni regolari **unione**, **concatenazione** e **star** come segue:

1. **Unione**: $A \cup B = \{x \mid x \in A \vee x \in B\}$
2. **Concatenazione**: $A \circ B = \{xy \mid x \in A \wedge y \in B\}$
3. **Star**: $A^* = \{x_1 x_2 \dots x_n \mid k \geq 0 \wedge \forall i \; x_i \in A\}$

L'operazione *star* è un'operazione unaria e non binaria come le altre due. Si applica quindi ad un solo linguaggio invece che a due. Funziona combinando le stringhe in A con se stesse per ottenere una stringa nel nuovo linguaggio. Poiché ogni numero include 0 come possibilità, la stringa vuota ε è sempre un membro di A^*, indipendentemente da cosa A sia.

Esempio. : sia Σ l'alfabeto composto dalle 26 lettere standard $\{a, b, \dots, z\}$. Se $A = \{\text{good}, \text{bad}\}$ e $B = \{\text{boy}, \text{girl}\}$ allora:

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \text{goodgoodgood}, \text{goodgoodbad}, \text{goodbadgood}, \text{goodbadbad}, \dots\}$$

Una collezione di oggetti si dice chiusa sotto un'operazione se applicando tale operazione ai membri della collezione l'oggetto ritornato è ancora nella collezione di partenza (es. $\mathbb{N}$ è chiuso rispetto al prodotto ma non rispetto alla divisione).

3.4 Chiusura

3.4.1 Rispetto all'unione

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione $\cup$.

Siano A_1, A_2 due linguaggi regolari, vogliamo dimostrare che $A_1 \cup A_2$ è un linguaggio regolare. Poiché A_1 e A_2 sono regolari, sappiamo che esiste un automa a stati finiti M_1 che riconosce A_1 ed un automa M_2 che riconosce A_2. Per dimostrare che $A_1 \cup A_2$ è regolare dimostriamo un automa a stati finiti M che riconosce $A_1 \cup A_2$.

Sia $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e sia $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, assumendo che abbiano lo stesso alfabeto Σ, costruiamo $M = (Q, \Sigma, \delta, q, F)$ come segue

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \wedge r_2 \in Q_2\}$. Questo insieme è il prodotto cartesiano degli insiemi $Q_1 \times Q_2$. È l'insieme delle coppie degli stati, la prima di Q_1 e la seconda di Q_2.
2. Σ, l'alfabeto, è lo stesso in M_1 e M_2. Assumiamo per semplicità che sia lo stesso.
3. δ, la funzione di transizione, è definita come segue. Per ogni $(r_1, r_2) \in Q$ e per ogni $a \in \Sigma$, sia

   $$\delta\left((r_1, r_2), a\right) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

   Quindi δ prende uno stato di M (che è una coppia di stati di M_1 e M_2) insieme ad un simbolo di input e ritorna il prossimo stato di M.
4. q_0 è la coppia (q_1, q_2).
5. F è l'insieme delle coppie nelle quali è accettato lo stato in M_1 oppure M_2. Possiamo scriverla come

   $$F = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$$

Questo conclude la costruzione dell'automa M che riconosce l'unione di A_1 e A_2.

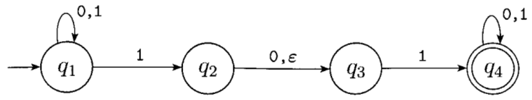
3.4.2 Rispetto alla concatenazione

La classe dei linguaggi regolari è chiusa rispetto all'operazione di concatenazione $\circ$.

Per dimostrare questo teorema dobbiamo anche questa volta costruire un nuovo automa, con la differenza che questa volta non accetterà l'input se lo accetta M_1 oppure M_2. M deve accettarlo se l'input può essere spezzato in due pezzi, dove M_1 accetta il primo pezzo e M_1 accetta il secondo pezzo. Il problema è che M non sa quando spezzare l'input. Per risolvere questo problema dobbiamo introdurre il concetto di *non determinismo*.

3.5 Non determinismo

Fino ad ora ogni passo di una computazione portava ad un unica via. Quando una macchina è in un dato stato e legge il prossimo simbolo di input, sappiamo quale sarà il prossimo stato. Questo meccanismo viene chiamato **deterministico**. In un sistema **non deterministico** scelte differenti possono esistere per il prossimo stato, in ogni punto.

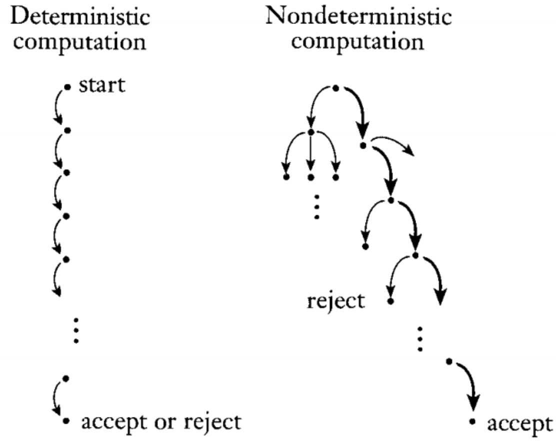


La differenza fra un automa a stati finiti deterministico (*DFA*, Deterministic Finite Automaton) e non deterministico (*NFA*, Nondeterministic Finite Automaton):

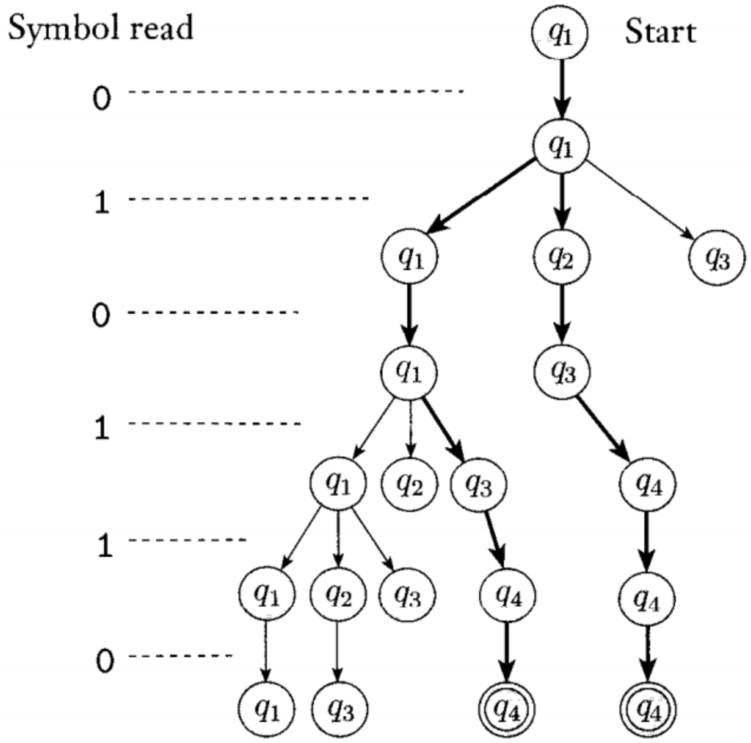
- Ogni stato di un DFA ha esattamente una freccia uscente per ogni simbolo dell'alfabeto. Un NFA vìola questa regola.
- Un stato di un NFA può avere zero, una o più frecce uscenti per ogni simbolo dell'alfabeto (compreso ε).

3.5.1 Come computa un NFA?

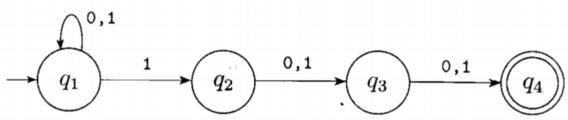
Supponiamo di trovarci in un NFA con una stringa in input che ci ha condotti allo stato q_1 che ha più modi di procedere. Per esempio, diciamo che il prossimo simbolo di input è 1. Dopo aver letto il simbolo la macchina si divide in copie multiple di se stessa e segue **tutte** le possibili vie in parallelo. Ogni copia della macchina prende una direzione e continua la computazione come prima, dividendosi a sua volta in più copie se necessario. Se il prossimo simbolo di input non compare in nessuna freccia uscente dallo stato corrente di ogni copia, quella copia cessa di esistere assieme al ramo della computazione a lei associato. Alla fine dell'input, se almeno una delle copie si trova in uno stato accettante, il NFA accetta la stringa di input.
Se viene incontrato uno stato con una freccia uscente con il simbolo ε, senza nemmeno leggere l'input la macchina se divide in copie multiple, una per ogni freccia uscente marcata da ε e una copia rimane ferma sullo stato corrente. Successivamente la macchina procede in modo non deterministico come prima.



Esempio. di computazione non deterministica dell'automa visto sopra con l'input 010110:



Esempio. : sia A il linguaggio che consiste in tutte le stringhe in $\{0, 1\}$ che contengono un 1 nella terza posizione dalla fine (es. 000100). Il seguente NFA a tre stati riconosce A:

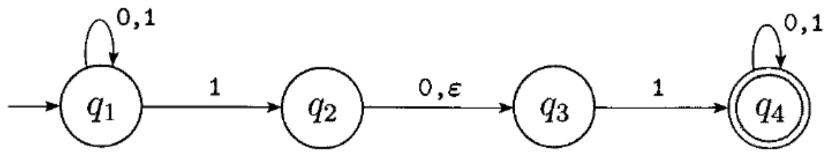


3.5.2 Definizione

Una NFA è una quantupla $(Q,\ \Sigma,\ \delta,\ q_0,\ F)$ dove:

1. Q è un insieme finito di stati
2. Σ è un insieme finito di simboli detto alfabeto
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$
4. $q_0 \in Q$ è lo stato iniziale
5. $F \subseteq Q$ è un insieme degli stati finali

Esempio.



L'automa può essere rappresentato formalmente mediante la seguente sintassi:

$$A = (\{q_1, q_2, q_3, q_4\},\ \{0, 1\},\ \delta,\ q_1,\ \{q_4\})$$

dove δ è definita come:

$\delta(q_1, 0) = \{q_1\}$	$\delta(q_2, 0) = \{q_3\}$	$\delta(q_3, 0) = \varnothing$	$\delta(q_4, 0) = \{q_4\}$
$\delta(q_1, 1) = \{q_1, q_2\}$	$\delta(q_2, 1) = \varnothing$	$\delta(q_3, 1) = \{q_4\}$	$\delta(q_4, 1) = \{q_4\}$
$\delta(q_1, \varepsilon) = \varnothing$	$\delta(q_2, \varepsilon) = \{q_3\}$	$\delta(q_3, \varepsilon) = \varnothing$	$\delta(q_4, \varepsilon) = \varnothing$

3.5.3 Accettazione di una stringa

Sia $N = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ un NFA. Diciamo che N accetta una stringa w nell alfabeto Σ se e solo se w può essere scritta nella forma $y_1, \ldots, y_n$ dove $\forall i \in [1, n] : y_i \in (\Sigma \cup \{\varepsilon\})$ ed esiste una sequenza di stati $R_0, \ldots, R_n \in Q$ tali che:

- R_0 è lo stato iniziale
- $R_m \in F$
- $\forall i \in [0, m-1] : R_{i+1} \in \delta(R_i, y_{i+1})$, ovvero che lo stato successivo (R_{i+1}) appartiene all'insieme degli stati ottenuti dalla funzione di transizione applicata allo stato corrente (R_i) e al prossimo carattere in input (y_{i+1}).

3.5.4 Equivalenza tra NFA e DFA

Dimostrare che ogni NFA ha un equivalente DFA.

Sia $N = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ un NFA che riconosce il linguaggio A, costruisco un DFA $M = (Q',\ \Sigma,\ \delta',\ q_0',\ F')$ che riconosce esattamente A. Assumiamo per semplicità che non vi siano $\varepsilon-$transizioni. Definiamo le componenti di M:

- $Q' = \mathcal{P}(Q)$, insieme delle parti di Q
- $\delta'(R, a) = \bigcup\limits_{r \in R} \delta(r, a), \quad R \in Q', a \in \Sigma$
- $q_0' = \{q_0\}$

- $F' = \{R \in \mathcal{P}(Q) \mid \exists r \in R, r \in F\}$, ovvero esiste un insieme R nell'insieme delle parti di Q tale che R contenga almeno uno stato accettante (finale) di N

Verifichiamo che le funzioni di transizione siano ben tipate:

1. NFA: $\delta : Q \times \Sigma \to \mathcal{P}(Q)$
2. DFA: $\delta' : \underbrace{\mathcal{P}(Q)}_{Q'} \times \Sigma \to \underbrace{\mathcal{P}(Q)}_{Q'}$

Generalizziamo ora il caso delle ε-transizioni definendo una funzione $E(R) = \{q \mid q \in Q\}$ può essere raggiunto da uno stato in R seguendo solamente ε-transizioni (anche zero). Riformulando i componenti visti sopra:

- $\delta'(R, a) = \bigcup\limits_{r \in R} E\left(\delta(r, a)\right)$
- $q_0 = E(\{q_0\})$

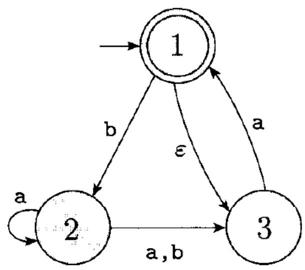
3.5.5 Linguaggi regolari

Andiamo a dimostrare che un linguaggio è regolare se e solo se esiste un NFA che lo riconosce. Dimostrazione:

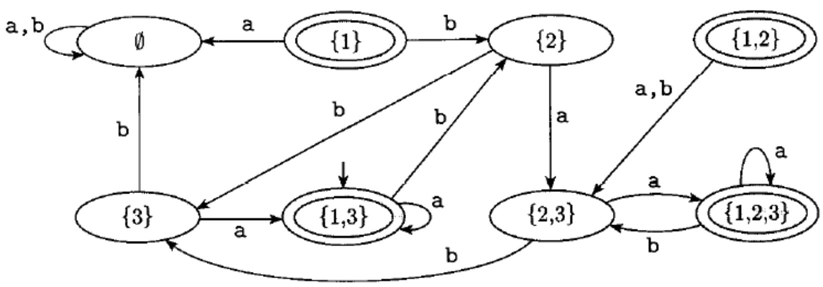
> $\Rightarrow$ Se un linguaggio è regolare, per definizione esiste un DFA che lo riconosce. Ma un DFA è un caso speciale di un NFA, quindi la prima parte è dimostrata.
>
> $\Leftarrow$ Sia A un linguaggio tale che A è riconosciuto da un NFA. Per il teorema esiste un DFA che riconosce A, quindi A è regolare.

Esempio. di conversione di un NFA in un DFA

NFA:

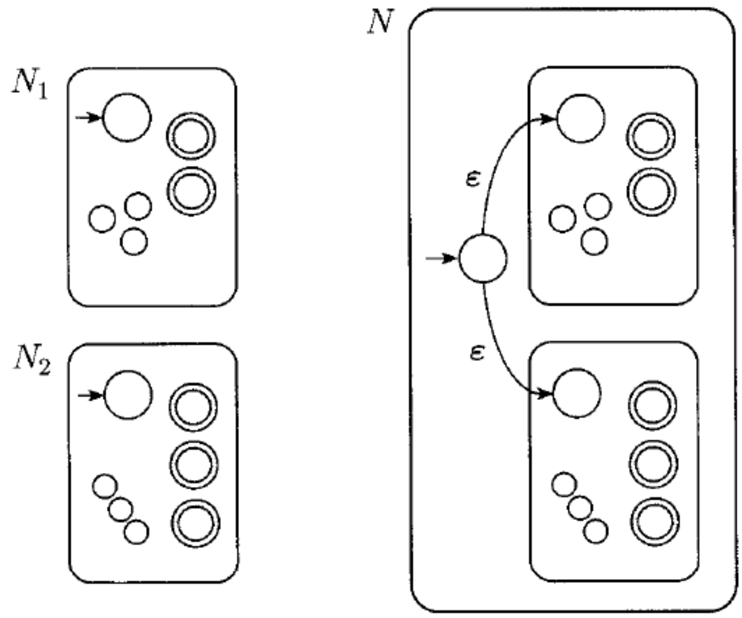


DFA:



Chiusura rispetto all'unione La classe dei linguaggi regolari è chiusa rispetto all'unione. Dimostrazione:

Siano A_1 e A_2 due linguaggi regolari, allora esistono due NFA N_1 ed N_2 che li riconoscono. Costruisco da N_1 ed N_2 un NFA che riconosce $A_1 \cup A_2$, da cui concludo che $A_1 \cup A_2$ è regolare.



Definizione formale:

$$N_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1) \qquad N_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2)$$

sia

$$O = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

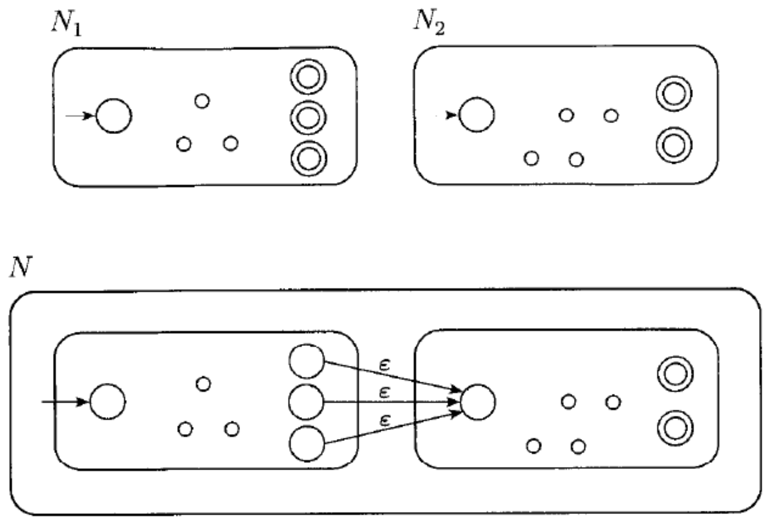
dove:

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $F = F_1 \cup F_2$
- e la funzione di transizione è definita come

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \\ \delta_2(q,a) & \text{se } q \in Q_2 \\ \{q_1, q_2\} & \text{se } q = q_0 \wedge a = \varepsilon \\ \varnothing & \text{se } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

Chiusura rispetto alla concatenazione La classe dei linguaggi regolari è chiusa rispetto alla concatenazione. Dimostrazione:

Siano A_1 e A_2 due linguaggi regolari, allora esistono due NFA N_1 ed N_2 che li riconoscono.



In pratica salto dagli stati finali (non più) del primo automa allo stato iniziale del secondo automa. Definizione formale:

$$N_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1) \qquad N_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2)$$

sia

$$N = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

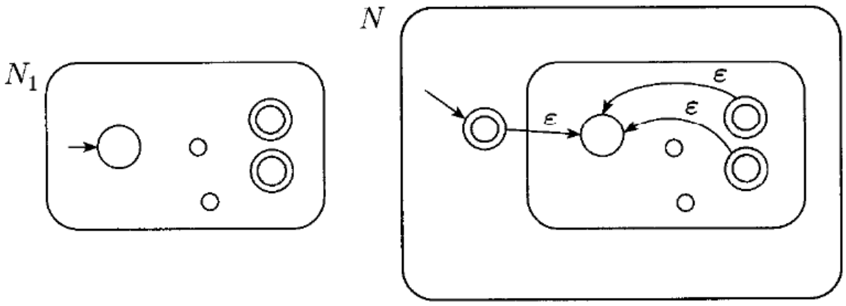
dove:

- $Q = Q_1 \cup Q_2$
- $F = F_2$
- e la funzione di transizione è definita come

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \land q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \land a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{se } q \in F_1 \land a = \varepsilon \\ \delta_2(q,a) & \text{se } q \in Q_2 \end{cases}$$

Chiusura rispetto all'operazione star La classe dei linguaggi regolari è chiusa rispetto all'operazione *star*. Dimostrazione:

Sia N_1 un linguaggio regolare, allora esiste un NFA N_1 che lo riconosce. Costruisco un automa $N = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ che accetta A^*:



Aggiungo un primo stato accettante e due ε-transizioni per poter iterare un numero arbitrario di volte. Il primo stato è accettante perche ε fa sempre parte dei A^*. Definizione formale:

$$N_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1)$$

sia

$$N = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

dove:

- $Q = Q_1 \cup \{q_0\}$
- $F = \cup\{q_0\}$
- e la funzione di transizione è definita come

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \land q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \land a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{se } q \in F_1 \land a = \varepsilon \\ \{\varepsilon\} & \text{se } q = q_0 \land a = \varepsilon \\ \varnothing & \text{se } q = q_0 \land a \neq \varepsilon \end{cases}$$

4 Espressioni regolari

Definiamo i linguaggi regolari in modo più compatto e semplice rispetto agli automi:

$$L((0 \cup 1)0^*) = \{w \mid w \text{ inizia con 0 oppure 1 e ha un numero arbitrario di } 0\}$$

Secondo le regole della precedenza $(^*, \circ, \cup)$, * è applicabile a 0 ma non a $(0 \cup 1)$.

4.1 Definizione

Diciamo che R è un espressione regolare su un alfabeto Σ se R è:

1. $a \mid a \in \Sigma$, ovvero se è un membro dell'alfabeto (che è sempre un'espressione regolare) $\quad L(a) = \{a\}$
2. ε $\quad L(\varepsilon) = \{\varepsilon\}$
3. $\varnothing$ $\quad L(\varnothing) = \{\varnothing\}$
4. $(R_1 \cup R_2)$, dove R_1 e R_2 sono espressioni regolari $\quad L((\overbrace{R_1}^{\text{regexp}} \cup R_2)) = \overbrace{L(R_1)}^{\text{linguaggio regolare}} \cup L(R_2)$
5. $(R_1 \circ R_2)$, dove R_1 e R_2 sono espressioni regolari $\quad L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$
6. (R_1^*), dove R_1 è un'espressione regolare $\quad L(R_1^*) = L(R_1)^*$

Esempio. in $\Sigma = \{0, 1\}$:

- $L(0^*10^*) = \{w \mid w \text{ contiene esattamente un } 1\}$
- $L(01 \cup 10) = \{01, 10\}$
- $L((0 \cup \varepsilon)1^*) = \{w \mid w \text{ è una stringa di soli 1 possibilmente preceduta da } 0\ \}$
- $L(1^* \circ \varnothing) = \varnothing$, concatenando l'insieme vuoto con qualunque altra cosa si ottiene insieme vuoto
- $L(\varnothing^*) = \{\varepsilon\}$, l'operatore di star su un insieme vuoto ritorna la combinazione di 0 stringhe, ovvero solamente stringa vuota

Identità delle espressioni regolari, dove R è una *regexp*:

- $R \cup \varnothing = R$ ✔
- $R \cup \varepsilon = R$ ✘
- $R \circ \varepsilon = R$ ✔
- $R \circ \varnothing = R$ ✘

4.2 Equivalenza con automi a stati finiti

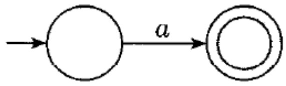
Teorema. *Un linguaggio è regolare se e solo se esiste un'espressione regolare che lo descrive. Quindi A è regolare se e solo se esiste una regexp in R tale che $L(R) = A$.*

Essendo un *se e solo se* vanno dimostrate entrambe le parti.

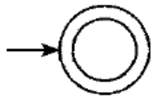
Lemma. *Se A è descritto da una rgexp, allora A è regolare.*

Sia A descritto da una regexp R. Per induzione sulla struttura di R procediamo per casi:

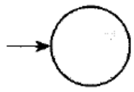
1) $R = a$ per qualche $a \in \Sigma$, Dimostro che $L(a)$ è regolare se trovo un NFA che lo riconosca:



2) $R = \varepsilon$. Allora $L(R) = \{\varepsilon\}$, e il seguente NFA riconosce $L(R)$



3) $R = \varnothing$. Allora $L(R) = \varnothing$, e il seguente NFA riconosce $L(R)$



4) $R = R_1 \cup R_2$

5) $R = R_1 \circ R_2$

6) $R = R_1^*$

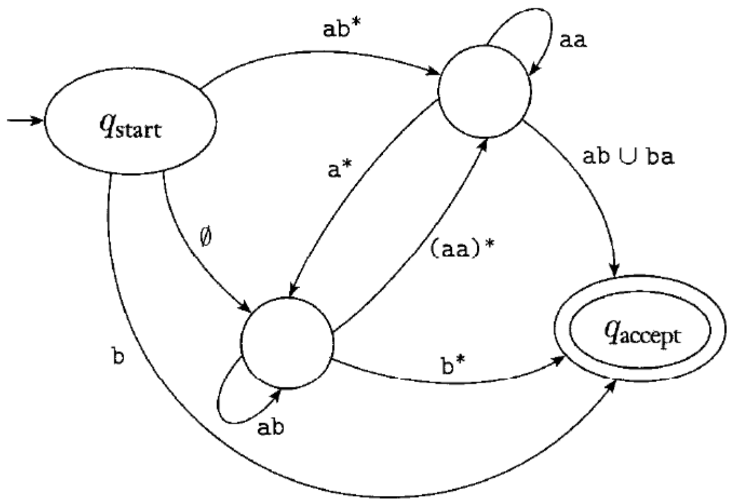
Per gli ultimi tre casi utilizziamo le costruzioni date delle dimostrazioni precedenti, ovvero che i linguaggi regolari sono chiusi rispetto alle operazioni regolari.

Lemma. *Se un linguaggio è regolare, allora esiste un'espressione regolare che lo riconosce.*

Per dimostrare questo lemma abbiamo bisogno di introdurre un nuovo tipo di automa, i GNFA, ovvero un automa a stati finiti non deterministico generalizzato, sui quali archi non vi sono più simboli, ma espressioni regolari.

4.3 GNFA

Esempio.



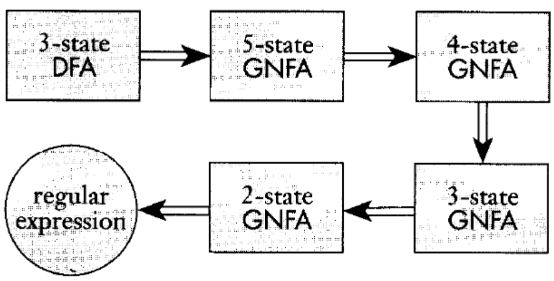
Ci focalizziamo su un GNFA con un formato particolare, ovvero

1. Lo stato iniziale ha archi uscenti verso tutti gli altri stati e nessun arco entrante
2. Lo stato finale ha archi entranti da tutti gli altri stati e nessun arco uscente
3. Per tutti gli altri stati esiste un arco verso ciascun altro stato, incluso se stesso, ma esclusi quelli finale ed iniziale

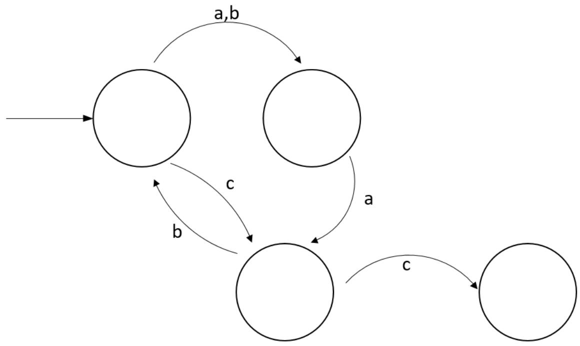
Per convertire un DFA in un GNFA ben formato è necessario seguire una serie di passaggi:

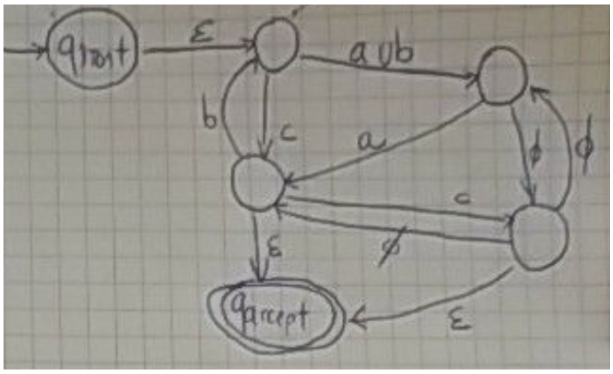
1. Aggiungi un nuovo stato iniziale con $\varepsilon-$transizioni verso lo stato iniziale originario (il nuovo stato iniziale non ha transizioni in entrata)
2. Aggiungi uno stato finale, metti $\varepsilon-$transizioni da ciascuno dei vecchi stati finali verso esso e rendi tali stati non finali
3. Se una freccia ha più etichette, rimpiazzale con l'unione delle etichette stesse $(a, b \Rightarrow a \cup b)$. Se una freccia è mancante, aggiungila ed etichettala con $\varnothing$.

Passaggi tipici di una conversione da GNFA a regular expression:

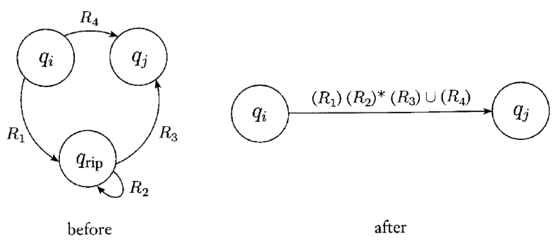


Esempio. di conversione da DFA a GNFA:

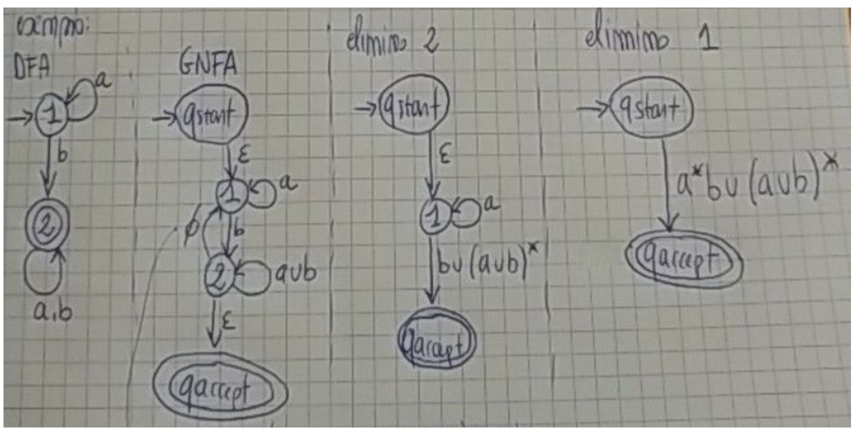




Esempio. di normalizzazione di un GNFA, che permette di ridurne gli stati al fine di ottenere un'espressione regolare:



Esempio. di normalizzazione di un GNFA:



5 Linguaggi non regolari

I linguaggi non regolari sono tutti quei linguaggi che non possono essere riconosciuti da nessun automa a stati finiti (DFA).

$A = \{0^n 1^n \mid n \geq 0\} \rightarrow$ non regolare

$A = \{w \mid w$ ha lo stesso numero di 0 e 1$\} \rightarrow$ non regolare

$A = \{w \mid w$ ha lo stesso numero di occorrenze delle sottostringhe 01 e 10$\} \rightarrow$ regolare

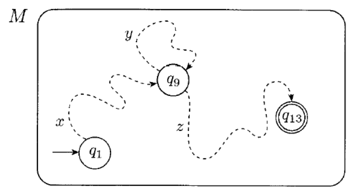
Andiamo ora a definire uno dei modi che permettono di determinare la regolarità di un linguaggio.

5.1 Pumping

Lemma. *Se A è un linguaggio regolare, esiste un numero p (pumping length) tale che ogni stringa di A di lunghezza almeno p, può essere divisa in tre pezzi xyz tali che*

1. $\forall i \geq 0 : xy^i z \in A$
2. $|y| > 0$
3. $|xy| \leq p$

Intuizione: Se A è un linguaggio regolare allora esiste un DFA M che lo riconosce. Imposto p al numero di stati di M. Prendiamo una stringa s tale che $|s| > p$. Poiché $|s| > p$ devo attraversare almeno $p+1$ stati per riconoscerla, da qui ricaviamo che uno stato dell'automa deve necessariamente essere attraversato almeno due volte.



Dimostrazione. Sia A un linguaggio regolare, allora esiste un DFA M che lo riconosce. Sia $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ e sia p il numero di stati in Q. Sia $s = s_1, \dots, s_n$ tale che $n \geq p$ e sia $r_1, \dots, r_{n+1}$ la sequenza di stati attraversati per riconoscere s, allora $r_{i+1} = \delta(r_i, s_i)$.
Tra i primi $p+1$ stati in $r_1, \dots, r_n$ deve esserci una ripetizione, chiamata prima occorrenza r_j e una seconda occorrenza chiamata r_l con $l \leq p+1$. Spezziamo s in tre sotto-stringhe xyz dove:

$x = s_1, \dots, s_{j-1}$

$y = s_j, \dots, s_{l-1}$

$z = s_l, \dots, s_n$

Verifichiamo ora le tre condizioni:

1. $\forall i \geq 0 : xy^i z \in A$
2. $|y| > 0$ vero perché considero occorrenze diverse dello stato
3. Sappiamo che $j \neq l$ e che $l \leq p$, da cui $|xy| \leq p$

$\square$

Per dimostrare che un linguaggio A è non regolare, posso dunque utilizzare il *puming lemma*:

1. Assumo per assurdo che A sia regolare
2. Poiché A è regolare, deve valere il *puming lemma*
3. Trovo una stringa s, con $|s| \geq p$ tale che per ogni particolare xyz, con $|xyz| > p$ ho che esiste i tale che $xy^i z \notin A$
4. Concludo che A non poteva essere regolare

Esempio. di applicazione del *puming lemma* per verificare la non regolarità di un linguaggio.

$$A = \{0^n 1^n \mid n \geq 0\}$$

Assumo per assurdo che A sia regolare, allora esiste p tale che ogni stringa $s \in A$ può essere scritta come xyz in modo che: 1) $\forall i \geq 0 : xy^i z \in A$, 2) $|y| > 0$ e 3) $|xy| \leq p$.

Prendo $s = 0^p 1^p$ e mostro che $xy^i z \notin A$ per qualche i. Procedo per casi:

1. y contiene solo 0. In questo caso $xy^2 z \notin A$ perché $xy^2 z$ contiene più 0 che 1.
2. y contiene solo 1. In questo caso $xy^2 z \notin A$ perché $xy^2 z$ contiene più 1 che 0.
3. y sia 0 che 1.

$$s = 00\underbrace{01}_{y}11$$

$$s = 00\underbrace{0101}_{y^2}11$$

In questo caso $xy^2 z \notin A$ perché $xy^2 z$ conterrà degli 1 prima di uno 0.

Esempio. di applicazione del *puming lemma* per verificare la non regolarità di un linguaggio.

$$C = \{w \mid w \text{ ha lo stesso numero di 0 e 1}\}$$

Sia $s = 0^p 1^p$. Poiché la parte della stringa che vado a pompare deve occorrere nei primi p caratteri di s, y deve contenere solo 0. Perciò $xy^2 z$ contiene più 0 che 1, da cui $xy^2 z \notin C$.

Esempio. di applicazione del *puming lemma* per verificare la non regolarità di un linguaggio.

$$F = \{ww \mid w \in \{0,1\}^*\}$$

Sia $s = 0^p 1 0^p 1$. Poiché $|xy| \leq p$, y deve contenere solo 0, perciò $xy^2 z$ contiene più 0 che 1, che è lecito, ma rompe la simmetria, da cui $xy^2 z \notin F$.

Esempio. di applicazione del *puming lemma* per verificare la non regolarità di un linguaggio.

$$D = \{1^{n^2} \mid n \geq 0\} \to \text{tutte le stringhe di 1 la cui lunghezza sia un quadrato perfetto}$$

Sia $s = 1^{p^2} = xyz$, allora $|s| = p^2$. Per il terzo punto del *pumping lemma* ($|xy| \leq p$) impostiamo:

$$|xy^2 z| = |xyz| + |y|$$

dove $|xyz| = p^2$, $|y| > 0$ e $|y| < p$. Quindi

$$|xy^2 z| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

perciò , poiché $|y| > 0$

$$p^2 < |xy^2 z| < (p+1)^2$$

ma allora $|xy^2 z|$ non è un quadrato perfetto, e $xy^2 z \notin D$.

6 Esercizi

Esercizio. *Dimostrare che la classe dei linguaggi regolari è chiusa rispetto alla concatenazione.*

Dimostrazione. Siano A e B due linguaggi regolari. Poiché sono regolari esisteno due DFA M ed N tali che $L(M) = A$ e $L(N) = B$. Sia $M = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e sia $N = (Q_2, \Sigma, \delta_2, q_2, F_2)$, definisco un DFA $O = (Q, \Sigma, \delta, q_0, F)$ dove:

1. $Q = Q_1 \times Q_2$
2. $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
3. $q = (q_1, q_2)$
4. $F = \{(q_1, q_2) \mid q_1 \in F_1 \wedge q_2 \in F_2\} = F_1 \times F_2$

□

Esercizio. *Dimostrare che $C = \{w \mid w$ ha lo stesso numero di 0 e 1$\}$ non è regolare, senza utilizzare il pumping lemma.*

Dimostrazione. Per assurdo assumo che C sia regolare. Poiché i linguaggi regolari sono chiusi rispetto all'intersezione, la seguente intersezione

$$C \cap 0^*1^*$$

dovrebbe essere regolare. Ma $C \cap 0^*1^* = \{01^n \mid n \geq 0\}$ e questo non è regolare, da cui C è non regolare. □

Esercizio. *Dimostrare che il linguaggio $D = \{w \mid w$ lo stesso numero di sottostringhe 01 e 10 $\}$ è regolare.*

Dimostrazione. Costruisco un DFA che non ho voglia di inserire... □

Esercizio. *Dimostrare che i linguaggi regolari sono chiusi rispetto all'operazione di complemento $\bar{A} = \{w \mid w \notin A\}$.*

Dimostrazione. Sia A un linguaggio regolare. Allora esiste un DFA $M = (Q, \Sigma, \delta, q_0, F)$ tale che $L(M) = A$. Costruisco un altro DFA $N = (Q, \Sigma, \delta, q_0, Q/F)$, e osservo che $L(N) = \bar{A}$. □

Esercizio. *Dimostrare che il linguaggio $B = \{w \mid w$ non contiene la stringa ab $\}$ è regolare.*

Dimostrazione. Costruisco un DFA N tale che $L(N) = \{w \mid w$ contiene la stringa ab $\}$. Poiché $B = L(\bar{N})$ concludo la chiusura rispetto al complemento che B è regolare. □

Esercizio. *Dimostrare che il linguaggio $E = \{0^i1^j \mid i > j\}$ è non regolare.*

Dimostrazione. Assumo per assurdo che E sia regolare, allora esiste p tale che ogni stringa $s \in E$ è tale che $|s| > p$ può essere scritta come xyz dove:

1. $\forall i xy^iz \in E$
2. $|y| > 0$
3. $|xy| \leq p$

Prendo $s = o^{p+1}1^p$. Poiché $|xy| \leq p$, so che y comprende solo 0. La stringa $xy^0z \notin E$, perché essa comprende meno di $p+1$ zeri:

$$\underbrace{\overbrace{0\ldots0}^{p+1}}_{x}\underbrace{0}_{y}\underbrace{\overbrace{1\ldots1}^{p}}_{z}$$

□

7 Context-free languages

In questo capitolo introdurremo le grammatiche *context-free*, un metodo più potente per descrivere linguaggi.

7.1 Context-free grammars

Il seguente è un esempio di una grammatica *context-free*, che chiameremo G_1:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

Una grammatica consiste in una collezione di *regole di sostituzione*, anche chiamate **produzioni**. Ogni regola appare come una linea e comprende un *simbolo* e una stringa, separati da una freccia. Il simbolo è chiamato **variabile**, mentre le stringhe consistono in altre variabili e simboli, chiamati **terminali**. I simboli delle variabili spesso sono rappresentate in lettere maiuscole, mentre i terminali sono analoghi all'alfabeto di input e sono spesso rappresentati da lettere minuscole, numeri o caratteri speciali. Una tra le variabili della grammatica è la **variabile di start**, e solitamente si trova a sinistra della regola più in alto.

La grammatica G_1 contiene dunque tre produzioni (le tre righe), le sue variabili sono A e B, dove A è la variabile di start. I sui terminali sono 0, 1 e #.

Si utilizza una grammatica per descrivere un linguaggio generando ogni stringa di quel linguaggio nel seguente modo:

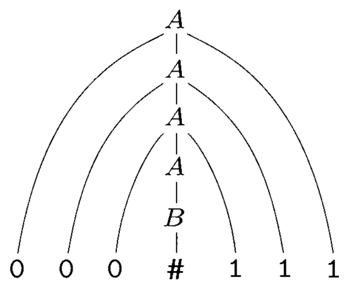
1. Si scrive la variabile di start
2. Si seleziona una della variabili disponibili e la si riscrive come parte destra di una produzione per quella variabile
3. Si torna al punto 2 fino a che ci sono ancora variabili disponibili

Per esempio la grammatica G_1 genera la stringa 000#111. La sequenza di sostituzioni per ottenere la stringa viene detta **derivazione**. La derivazione della stringa 000#111 nella grammatica G_1 é

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Tutte le stringhe generate in questo modo costituiscono il *linguaggio della grammatica.* Indichiamo con $L(G_1)$ il linguaggio della grammatica G_1, che è formalmente definito come $\{0^n\#1^n \mid n \geq 0\}$. Ogni linguaggio che può essere generato da una grammatica *context-free* è chiamato *context-free language (CFL).*

Un altro modo per indicare visivamente una grammatica *context-free* sono i *parse tree*



dove la radice rappresenta la start variable, le foglie i terminali e i nodi intermedi sono dettati dalle produzioni.

7.1.1 Definizione

Una grammatica context-free è una 4-tupla $(V,\ \Sigma,\ R,\ S)$ dove

1. V è un insieme finito di variabili
2. Σ è un insieme finito, disgiunto da V, di terminali
3. R è un insieme finito di regole (produzioni), dove ogni produzione ha la forma $A \to w$ dove $A \in V$ e $w \in (\Sigma \cup V)^*$, ovvero che w può essere una stringa di terminali e/o variabili
4. $S \in V$ è la start variable

Siano $u, v, w \in (\Sigma \times V)^*$ e sia $A \to w$ una produzione. Diciamo che uAv produce uwv ($uAv \Rightarrow uwv$). Diciamo che u deriva da v ($u \overset{*}{\Rightarrow} v$) sse $u = v$ oppure $\exists w_1, \dots, w_n \mid u \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n \Rightarrow v$.

Data una grammatica $G = (V,\ \Sigma,\ R,\ S)$, definiamo il linguaggio generato da G come l'insieme delle stringhe di terminali derivabili dalla start variable:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

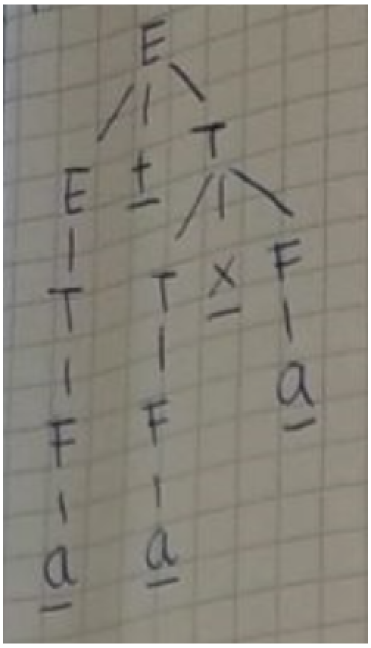
Esempio. Dato $G = (\{S\}, \{a, b\}, R, S)$, dove R è definito da $S \to aSb \mid SS \mid \varepsilon$, determinare alcune stringhe che può comporre.

- $abab:$ $\quad S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$
- $aabb:$ $\quad S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

Supponendo quindi che a rappresenti la parentesi sinistra "(" e b quella destra ")", il linguaggio di G è quello di tutte le stringhe le cui parentesi siano annidate correttamente.

Esempio. Generare il *parse tree* che generi $a + a \times a$ dati

- $E \Rightarrow E + T \mid T$
- $T \Rightarrow T \times F \mid F$

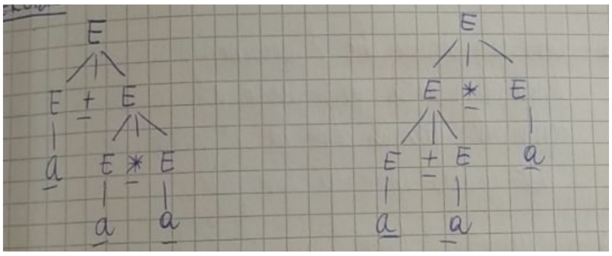


7.1.2 Ambiguità

Una grammatica è *ambigua* se e solo se esiste $w \in L(G)$ con almeno due *leftmost* diversi. Per *leftmost* si s'intende che nel processo di derivazione (e di conseguenza nel *parse tree*) scrivo il terminale a sinistra.

Esempio. Dato $E \to E + E \mid E \times E \mid a \mid (E)$, verificare che sia ambiguo.

A questo punto sviluppo due *parse tree*, e dal momento che ottengo la stessa stringa "$a + a \times a$" in due modi differenti, concludo che la grammatica è ambigua.



7.2 Forma normale di Chomsky

Un *context-free language* è in forma normale di Chomsky se ogni sua produzione ha la forma $A \to BC$ oppure $A \to a$ con l'eccezione che B, C non possono essere la variabile di start. Inoltre ammettiamo $S \to \varepsilon$ dove S è la start variable.

Teorema. *Ogni linguaggio context-free è generabile da una grammatica context-free in forma normale di Chomsky. [vedi pag 99, theorem 2.6]*

Dimostrazione. Definiamo un algoritmo per convertire una CFG qualsiasi in forma normale:

1. Sia S lo start symbol, invento un nuovo start symbol $S' \to S$
2. Sia $A \to \varepsilon$ una ε-produzione con $A \neq S$. Allora cancello $A \to \varepsilon$ e in tutti i punti in cui A occorre, aggiungo una nuova produzione dove A è cancellata per ogni possibile occorrenza di A, ovvero

$$\begin{aligned} R \to uAwAv \rightsquigarrow & R \to uAwAv \text{ non cancello occorrenze di } A \\ & R \to uwAv \text{ cancello 1 occorrenza di } A \\ & R \to uAwv \text{ cancello 2 occorrenze di } A \\ & R \to uwv \text{ cancello 3 occorrenze di } A \end{aligned}$$

   Se avessi

$$\begin{aligned} R \to A \rightsquigarrow & R \to A \\ & R \to \varepsilon \end{aligned}$$

3. Elimina le produzioni unitarie della forma $A \to B$ se esiste $A \to B$ e $B \to w_1 \mid \ldots \mid w_n$, cancella $A \to B$ e sostituiscilo con $A \to w_1 \mid \ldots \mid w_n$.
4. A questo punto tutte le produzioni della grammatica hanno almeno due simboli a destra.
5. Nelle regole precedenti rimpiazzo ogni terminale u_i con $1 \leq i \leq k$ con un non terminale U_i e aggiungi $U_i \to u_i$

□

Esempio. Normalizzazione il CFG G nella forma normale di Chomsky.

1. Il CFG è mostrato sulla sinistra. Il risultato dell'applicazione del primo step per creare un nuovo start symbol appare sulla destra.

$$\begin{aligned} A &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned} \qquad \begin{aligned} S_0 &\to S \\ A &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned}$$

2. Rimuovendo le ε-produzioni $B \to \varepsilon$ sulla sinistra, e $A \to \varepsilon$ sulla destra:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \\ A &\to B \mid S \mid \varepsilon \\ B &\to b \mid \varepsilon \end{aligned} \qquad \begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A &\to B \mid S \mid \varepsilon \\ B &\to b \end{aligned}$$

3. Rimuovendo le produzioni unitarie $S \to S$, sulla sinistra e $S_0 \to S$ sulla destra:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A &\to B \mid S \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to S \mid ASA \mid aB \mid a \mid SA \mid AS \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to B \mid S \\ B &\to b \end{aligned}$$

4. Rimuovendo le produzioni unitarie $A \to B$ e $A \to S$:

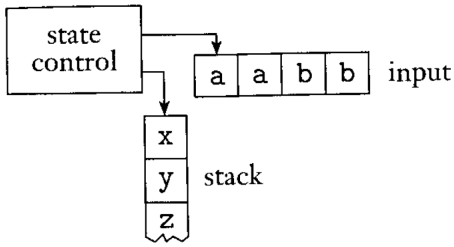
$$\begin{aligned} S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to B \mid S \mid b \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to S \mid b \mid ASA \mid aB \mid a \mid SA \mid AS \\ B &\to b \end{aligned}$$

5. Convertiamo le produzioni rimanenti nella forma corretta aggiungendo le variabili e le produzioni necessarie. Il risultato finale è

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ S &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ A &\to B \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ A_1 &\to SA \\ U &\to a \\ B &\to b \end{aligned}$$

8 Push-down automata

In questa sezione introdurremo i *pushdown automata.* Questi automi sono simili agli automi finiti non deterministici, ma hanno un componente in più chiamato stack. Lo stack fornisce memoria addizionale a quella finita disponibile ni NFA. Lo stack permette dunque a questi speciali automi di riconoscere alcuni linugaggi non regolari. Gli autocmi *pushdown* sono equivalenti in espressività alle grammatiche *context-free.*



Notiamo che l'alfabeto dello stack e di input possono essere differenti.

8.1 Definizione

Un automa *pushdown* è una 6-tupla $(Q,\ \Sigma,\ \mathit{\Gamma},\ \delta,\ q_0,\ F)$ dove

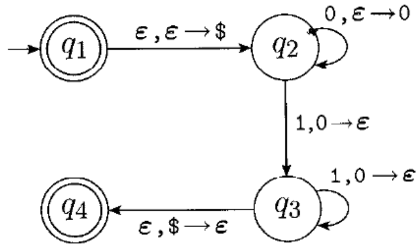
1. Q è un insieme finito di stati
2. Σ è un insieme finito di simboli di input
3. $\mathit{\Gamma}$ è un insieme finito di simboli per lo stack
4. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\mathit{\Gamma} \cup \{\varepsilon\}) \to \mathcal{P}(Q \times (\mathit{\Gamma} \cup \{\varepsilon\}))$
5. $q_0 \in Q$ è lo stato iniziale
6. $F \subseteq Q$ è l'insieme degli stati finali

Un PDA $M = (Q,\ \Sigma,\ \mathit{\Gamma},\ \delta,\ q_0,\ F)$ accetta l'input w se w si può scrivere come $w_1 w_2 ... w_n$ dove $\forall i\ w_i \in \Sigma \cup \{\varepsilon\}$ (ovvero come negli NFA posso avere ε-transizioni spontanee) ed esiste una sequenza di stati $r_0, r_1, \ldots, r_n \in Q$ e stack $s_0, s_1, \ldots, s_n \in \mathit{\Gamma}^*$ tali che:

1. $r_0 = q_0$ e $s_0 = \varepsilon$, ovvero che M inizia nello stato iniziale e con lo stack vuoto
2. $r_n \in F$, ovvero che M finisce di computare in uno stato accettante dopo aver concluso di leggere l'input
3. $\forall i \in [0, n-1]$: $(r_{i+1}, b) \in \delta(r_1, w_{i+1}, a)$ dove $s_i = at$ e $s_{i+1} = bt$ per qulche $a, b \in \mathit{\Gamma} \cup \{\Sigma\}$ e $t \in \mathit{\Gamma}^*$. Ovvero che per tutti gli stati, lo stato successivo (r_{i+1}) è calcolato dallo stato precedente e dal successivo simbolo di input $(\delta(r_i, w_{i+1}))$. Però in ogni momento posso leggere un simbolo dallo stack e aggiornare lo stato dello stack. Quindi valuto sia chi transita dallo stato sia il carattere a in cima allo stack dello stato corrente nel tempo i (enllo stato r_i). Nello stato successivo voglio aggiornare a con b (pop di a e push di b). Però non vengono sempre fatte push e pop perché possono esserci anche le ε-transizioni e al posto di a e b possono esserci tre casi:
   (a) $a = \varepsilon, b \neq \varepsilon$: push
   (b) $a \neq \varepsilon, b = \varepsilon$: pop
   (c) $a = \varepsilon, b = \varepsilon$: non modifico lo stack, ma faccio un ε-transizione

Esempio. Trovare un PDA che riconosca il linguaggio $\{0^n 1^n \mid n \geq 0\}$.

L'idea è che se leggo 0 riempio lo stack, mentre leggendo 1 lo svuoto. Mantengo un simbolo di flag (\$) che mi indica quando lo stack è vuoto, e quando arrivo alla fine dell'input se lo stack è vuoto significa che l'input viene riconosciuto.

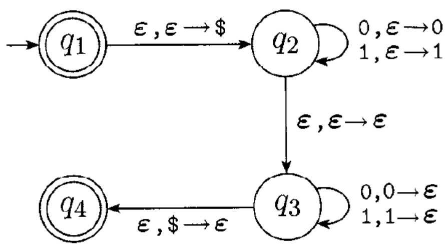


Quando scriviamo $a, b \to c$ significa che quando l'automa sta leggendo un a dall'input, può rimpiazzare il simbolo b nella testa dello stack con il simbolo c. Sia a che b e c possono essere ε. Se $a = \varepsilon$, l'automa esegue la transizione senza leggere nessun simbolo dall'input. Se $b = \varepsilon$ l'automa può eseguire la transizione senza leggere dall'input e prendere il primo simbolo dallo stack. Se $c = \varepsilon$ l'automa non scrive nessun simbolo nello stack eseguendo questa transizione.

Esempio. Determinare il PDA che riconosce il linguaggio

$\{ww^R \mid w \in \{0,1\}^*\}$ dove w^R equivale ad w rovesciato (reverse)

Il DFA deve riconoscere quindi, ad esempio, la stringa $\underbrace{100}_{w}\overbrace{001}^{w^R}$



Teorema. *Un linguaggio è context-free se e solo se è riconosciuto da un PDA.*

Dimostrazione. Se un linguaggio è *context-free* allora è riconosciuto da un PDA. Intuitivamente, se esiste un linguaggio è *context-free*, esiste una grammatica G che lo genera. Diamo una definizione informale del PDA:

1. Metti sullo stack "\$" seguito da S, dove S è lo *start symbol* della grammatica
2. Ripeti i seguenti passi
   (a) Se in cima allo stack ho A, scegli una produzione $A \to w$ e sostituisci A con w
   (b) Se in cima allo stack ho a, confrontalo con il prossimo carattere in input e fai la pop se sono uguali, fallisci termina questo ramo del non determinismo
   (c) Se in cima allo stack ho "\$" ed ho finito l'input, accetta

Per semplificare la definizione che andremo a vedere, occorre introdurre una notazione sharthand, più compatta, che ci permette di fare il push di stringhe intere anziché di singoli caratteri.

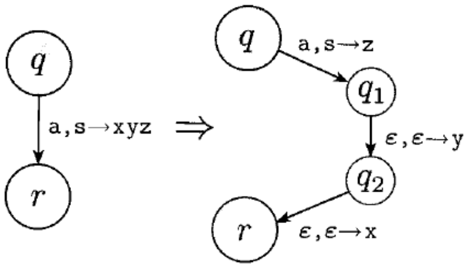


Figura 1: Implementazione della shorthand $(r, xyz) \in \delta(q, a, s)$

Sia l'immagine lo schema che ci permette di fare il push di una stringa e sia E l'insieme degli stati ausiliari utilizzati per simulare le transizioni necessarie per pushare una stringa e non un carattere.

Sia $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$ e sia definita la funzione di transizione come

$$\delta(q_{start}, \varepsilon, \varepsilon) = \{q_{loop}, S\$\}$$
$$\delta(q_{loop}, \varepsilon, A) = \{q_{loop}, w \mid A \to w \text{ è una produzione}\}$$
$$\delta(q_{loop}, a, a) = \{q_{loop}, \varepsilon\}$$
$$\delta(q_{loop}, \varepsilon, \$) = \{q_{accept}, \varepsilon\}$$

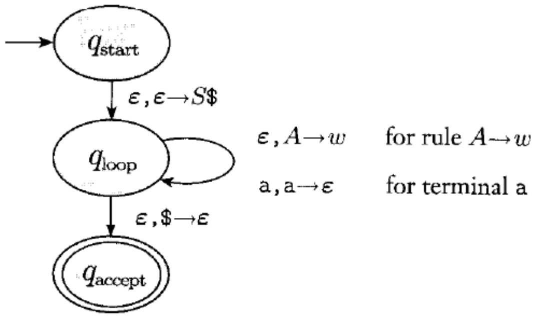


Figura 2: Diagramma di stato di Q

□

Dimostrazione. Se un li linguaggio è riconoscibile da un PDA, allora è *context-free.* Ci focalizziamo su un PDA con tre caratteristiche:

1. Ha un unico stato finale q_{accept}
2. Svuota lo stack prima di accettare
3. Ogni transizione è o una push o una pop, ma mai entrambe contemporaneamente

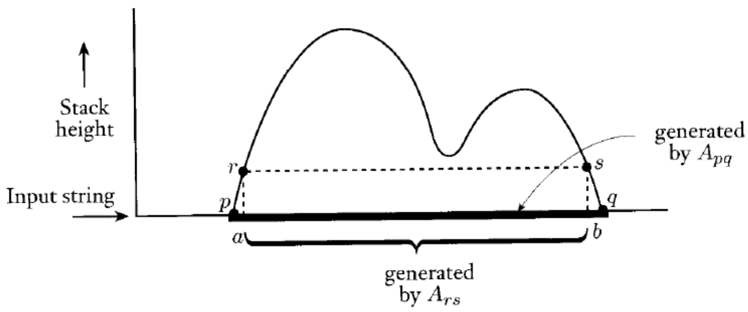
L'idea è che per ogni coppia di stati p e q del PDA, invento una variabile A_{pq} che genera tutte le stringhe che portano il PDA da p a q con lo stack vuoto. Quindi dato un input x il PDA farà come prima mossa una push e come ultima mossa una pop, dal momento che inizio e termino con lo stack vuoto.

- Se il carattere che poppo alla fine è lo stesso che ho pushato all'inizio, lo stack non si è mai svuotato. Simulo con

$$
\begin{aligned}
a &= \text{primo carattere letto} \\
b &= \text{ultimo carattere letto} \\
r &= \text{stato dopo di } p \\
s &= \text{stato prima di } q
\end{aligned}
$$

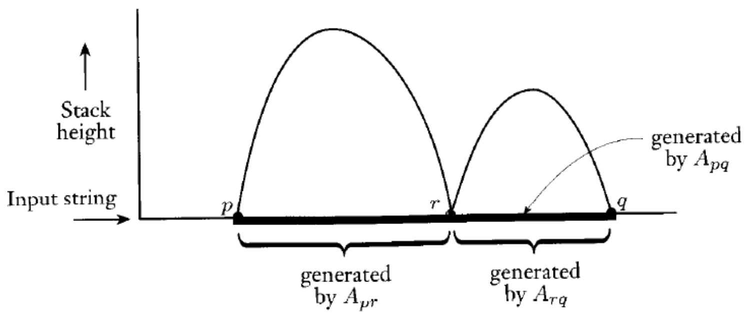
  allora

  $$A_{pq} \to aA_{rs}b$$



- Altrimenti esiste uno stato r dove lo stack si è svuotato. Simulo con:

  $$A_{pq} \to A_{pr}A_{rq}$$



□

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ un PDA delle forma specificata. Le variabili della grammatica sono

$$\{A_{pq} \mid p, q \in Q\}$$

mentre le produzioni sono

1. Per ogni $p, q, r, s \in Q$ e per ogni $a, b \in \Sigma \cup \{\varepsilon\}$, se $(r, u) \in \delta(p, a, \varepsilon)$ e $(q, \varepsilon) \in \delta(s, b, u)$
2. Per ogni $p, q, r, s \in Q$, aggiungi $A_{pq} \to A_{pr}A_{rq}$
3. Per ogni $p \in Q$ aggiungi $A_{pp} \to \varepsilon$