Calcolabilità e Linguaggi Formali

Giacomo De Liberali

28 settembre 2017

Indice

1 Introduzione

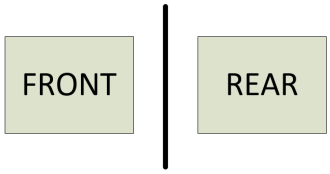
21 Settembre 2017

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

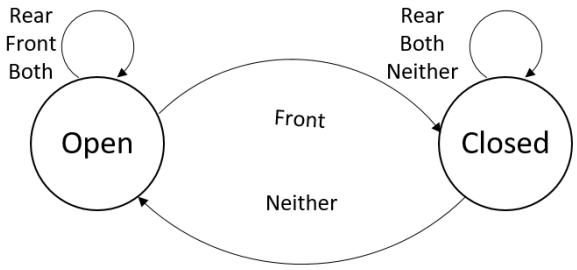
Il modello di computazione più elementare sono gli automi a stati finiti. Andiamo a rappresentare tramite un automa una porta con una bilancia da ogni lato che permetta l'accesso alle persone e impedisca alla porta stessa di colpire qualcuno:



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:



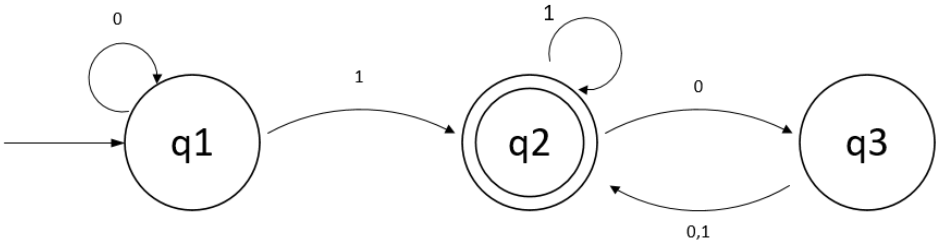
3 Automa a stati finiti

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:

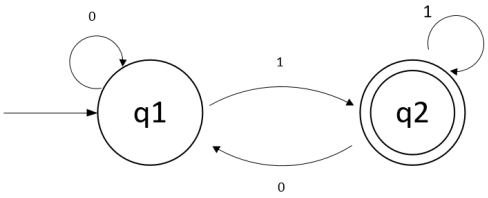
$$\begin{array}{ll} \delta(q_1, 0) = q_1 & \delta(q_1, 1) = q_2 \\ \delta(q_2, 0) = q_3 & \delta(q_2, 1) = q_2 \\ \delta(q_3, 0) = q_2 & \delta(q_3, 1) = q_2 \end{array}$$

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

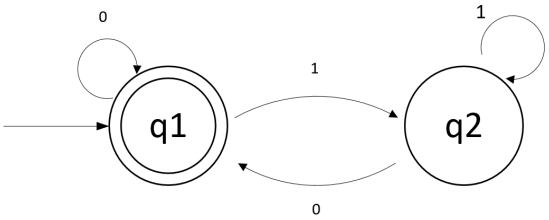
Ad esempio, la stringa 1101 è una stringa valida in quanto alla fine dell'input l'automa è nello stato q_2 che è l'unico stato finale.

➔ Esempio



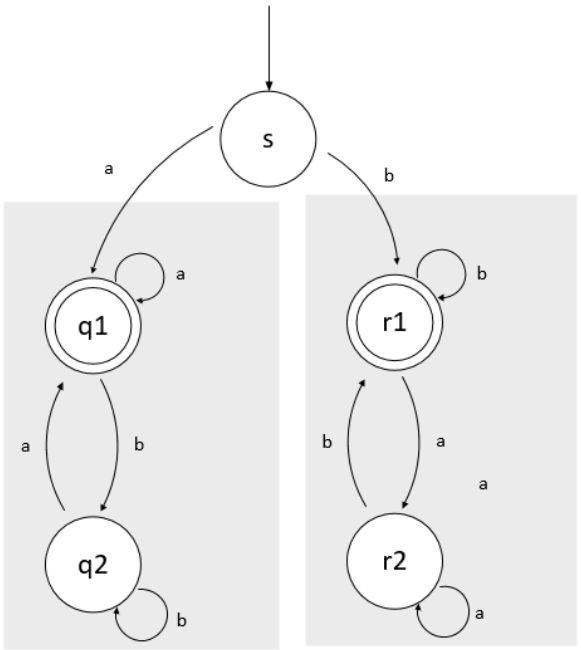
Questo automa riconosce stringhe in $\{0, 1\}$ che finiscono per 1. Nel caso di stringa vuota (ε) non entro in nessuno stato e non mi muovo.

➔ Esempio



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

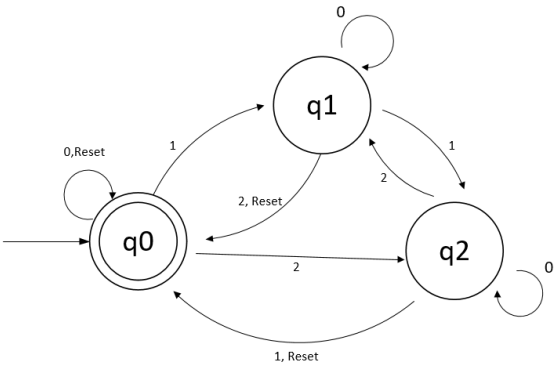
➔ Esempio



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

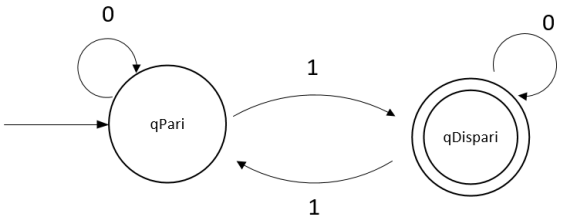
1. $R_0 = q_0$ (partendo dal nodo iniziale)
2. $R_n \in F$ (terminando in uno stato finale)
3. $\forall i \in [0, n-1] : \sigma(R_i, w_{i+1}) = R_{i+1}$ (per ogni input la *funzione di transizione* termini in uno stato finale)

M riconosce il linguaggio A se e solo se

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

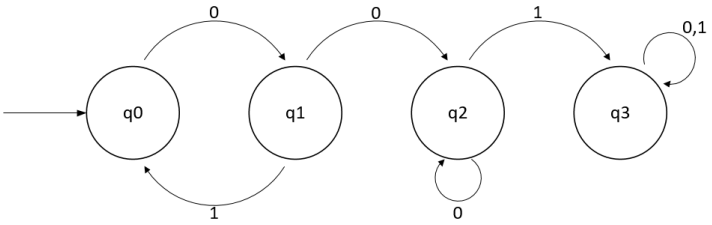
Un linguaggio A è *regolare* se e solo se esiste un automa a stati finiti M tale che M riconosce A.

➲ 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.

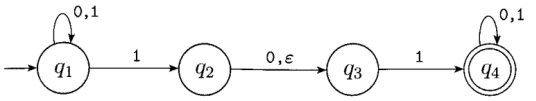
3.4.2 Rispetto alla concatenazione

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.

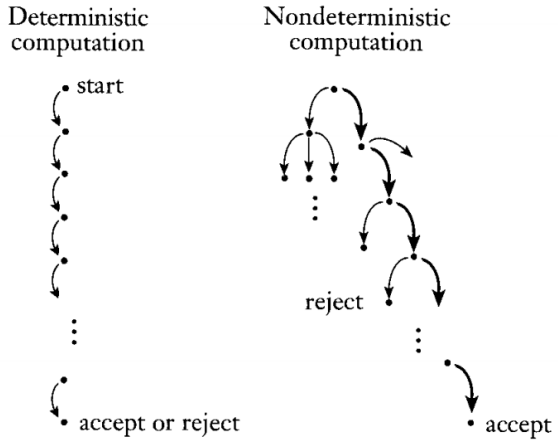


La differenza fra un automa a stati finiti deterministico (*DFA*, Deterministic Finite Automaton) e non deterministico (*NFA*, Nondeterministic Finite Automaton):

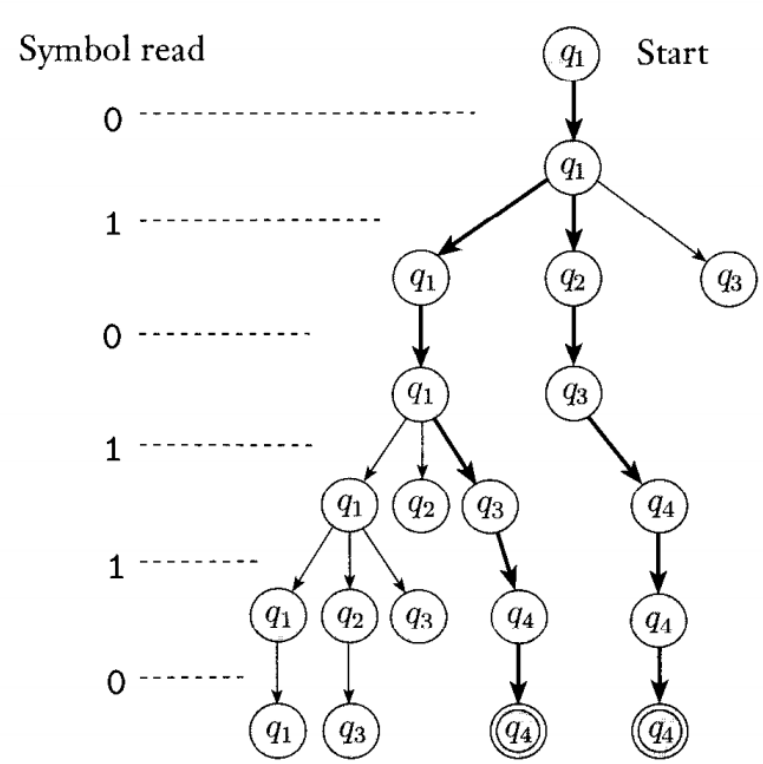
- Ogni stato di un DFA ha esattamente una freccia uscente per ogni simbolo dell'alfabeto. Un NFA vìola questa regola.
- Un stato di un NFA può avere zero, una o più frecce uscenti per ogni simbolo dell'alfabeto (compreso ε).

3.5.1 Come computa un NFA?

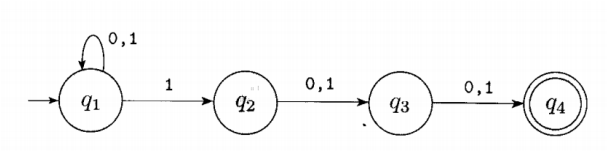
Supponiamo di trovarci in un NFA con una stringa in input che ci ha condotti allo stato q_1 che ha più modi di procedere. Per esempio, diciamo che il prossimo simbolo di input è 1. Dopo aver letto il simbolo la macchina si divide in copie multiple di se stessa e segue **tutte** le possibili vie in parallelo. 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:

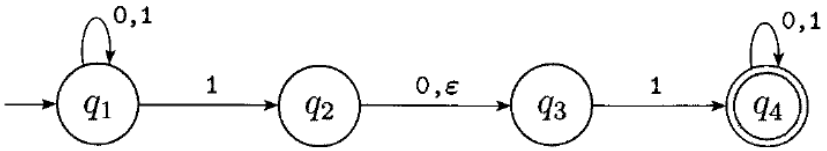


3.5.2 Definizione

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
- $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a), \quad R \in Q', a \in \Sigma$
- $q_0' = \{q_0\}$

- $F' = \{R \in \mathcal{P}(Q) \mid \exists r \in R, r \in F\}$, 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:

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

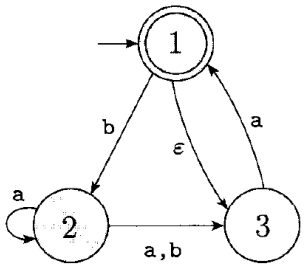
3.5.5 Linguaggi regolari

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

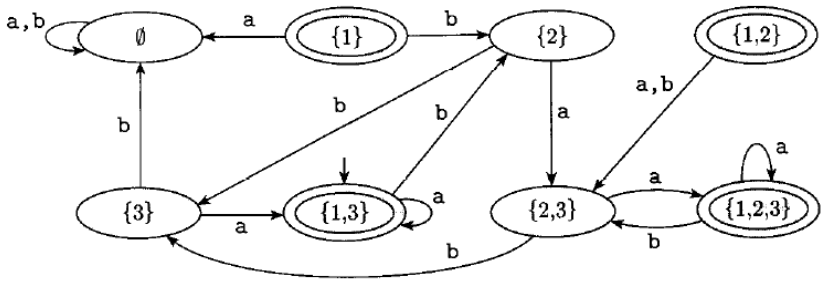
> $\Rightarrow$ Se un linguaggio è regolare, per definizione esiste un DFA che lo riconosce. Ma un DFA è un caso speciale di un NFA, quindi la prima parte è dimostrata.
>
> $\Leftarrow$ Sia A un linguaggio tale che A è riconosciuto da un NFA. Per il teorema esiste un DFA che riconosce A, quindi A è regolare.

Esempio di conversione di un NFA in un DFA

NFA:

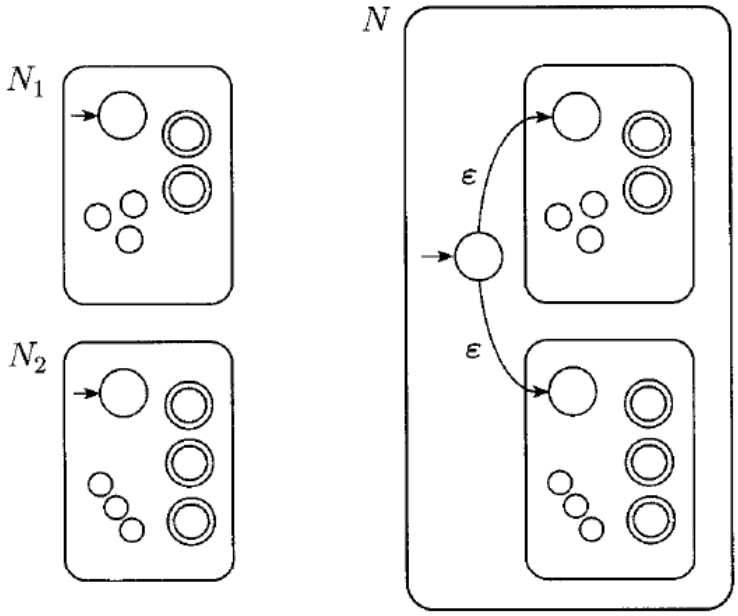


DFA:



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

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

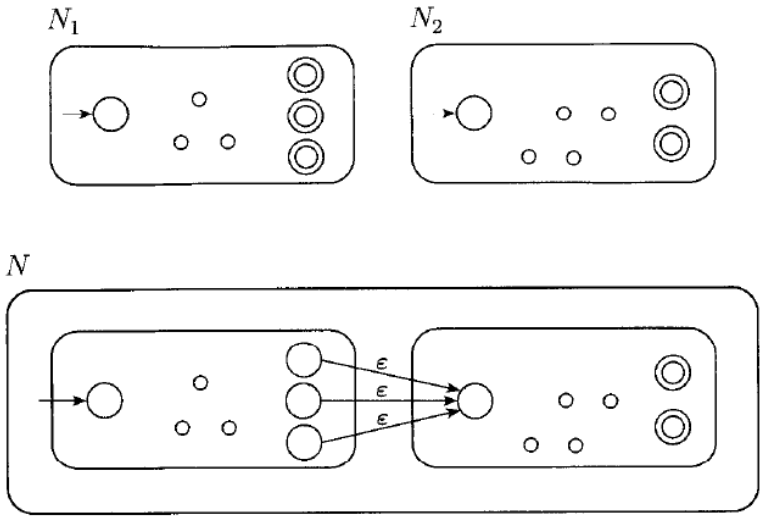
dove:

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $F = F_1 \cup F_2$
- 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

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