

Indice

1	Introduzione alla materia	2
1.1	Cardinalità degli insiemi	2
1.2	Alcune notazioni	3
1.3	Teorema di Cantor (1874)	4
1.4	Problema decisionale e ipotesi del continuo	5
2	Linguaggi regolari ed automi a stati finiti	6
2.1	Alfabeti e Linguaggi	6
2.2	Operazioni sui linguaggi	7
2.3	Automa a stati finiti	9
2.3.1	Automi deterministici (DFA)	11
2.3.2	Esempio esercizio (automi deterministici)	12
2.3.3	Automi non-deterministici (NFA)	14
2.3.4	Teorema Rabin-Scott (1959)	15
2.3.5	Esempio esercizio (automi non-deterministici)	16

1 Introduzione alla materia

Prima di iniziare con la presentazione di alcuni concetti fondamentali, si definisce l'**invariante induttiva**: pensando a qualsiasi linguaggio di programmazione, una generica condizione è sempre vera prima, durante e dopo un ciclo. Questo concetto ritornerà in futuro.

1.1 Cardinalità degli insiemi

Il motivo dell'interesse di un ripasso di un argomento trattato in passato è giustificato dal fatto che i dati manipolati in informatica sono (e)numerabili, ovvero è possibile metterli in corrispondenza biunivoca con i numeri naturali, essendo essi stessi rappresentati da numeri (binari).

Di seguito viene mostrato un richiamo ai concetti di base in relazione alla cardinalità di insiemi:

- ★ **Cardinalità.** Se S è un insieme, la sua *cardinalità* si rappresenta con il simbolo $|S|$.
- ★ **Equipotenza.** Due insiemi A e B sono *equipotenti* se esiste una funzione biiettiva del tipo $f : A \rightarrow B$ (cioè una funzione sia iniettiva che suriettiva; approfondimento: [link](#), oppure qui di seguito).
La *rappresentazione* matematica è la seguente $A \approx B$.
La relazione $|A| \leq |B|$ è possibile se esiste una funzione iniettiva $f : A \rightarrow B$. Si osservi che la funzione f stabilisce una corrispondenza tra gli elementi dei due insiemi. Infatti, l'**iniettività** assicura che la corrispondenza è stabilita elemento per elemento, mentre la **suriettività** assicura che la quantità degli oggetti nei due insiemi coincide.
- ★ **Insiemi finiti e infiniti.** Negli *insiemi finiti*, la cardinalità è un numero naturale corrispondente al numero di oggetti contenuti nell'insieme. Invece, negli *insiemi infiniti* la $|A|$ rappresenta la collezione degli insiemi Y tale che $Y \approx A$. Questa collezione viene chiamata **cardinalità** di A . Quindi, è vero che se $A \subseteq B$ allora si deduce che $|A| \leq |B|$.
- ★ **Insieme numerabile.** Un insieme A viene detto *numerabile* se è finito o equipotente all'insieme dei numeri naturali \mathbb{N} (ovvero, $A \approx \mathbb{N}$).
La cardinalità degli insiemi infiniti numerabili è denotata con \aleph_0 .
Un insieme A è finito se $|A| < \aleph_0$. Quindi, un insieme è numerabile se $|A| \leq \aleph_0$ (ovvero se è finito, quindi minore, oppure se è un insieme infinito numerabile rappresentato come \aleph_0 , quindi uguale).

1.2 Alcune notazioni

Se un generico *linguaggio di programmazione* viene indicato con la lettera \mathcal{L} e un generico *algoritmo* di un programma viene indicato con la lettera A , allora se un *algoritmo viene implementato in un linguaggio di programmazione*, è possibile scrivere la notazione insiemistica $A \in \mathcal{L}$. In un linguaggio di programmazione è possibile scrivere infiniti programmi, ovvero l'insieme dei numeri naturali \mathbb{N} .

Esistono due tipi di *rappresentazioni*:

- ☛ **Rappresentazione intensionale.** Rappresenta solo l'algoritmo, più nello specifico solamente quella specifica parte di codice (esempio a fine elenco).
- ☛ **Rappresentazione estensionale.** Rappresenta l'insieme ma tramite una forma più estesa (esempio a fine elenco).

L'*esecuzione* di un determinato algoritmo si indica con delle parentesi quadre più spesse $\llbracket A \rrbracket$. Quindi, la sua *rappresentazione intensionale* è solamente A , mentre la sua *rappresentazione estensionale* è data da $\llbracket A \rrbracket(i) = o$ (i è input e o è output). La rappresentazione estensionale può essere anche nel seguente modo $\llbracket M \rrbracket \in \{f \mid f = \llbracket M \rrbracket\}$ con $f = \{(x, f(x)) \mid x \in \mathbb{N}\}$.

Un programma restituisce uno o più risultati come numeri naturali \mathbb{N} , prendendo in input dei numeri naturali \mathbb{N} . Quindi, più formalmente si può scrivere $\mathbb{N} \rightarrow \mathbb{N}$. Questa rappresentazione non è altro che la definizione dei *problem* esistenti. Difatti, l'informatica si pone il dubbio che esista una certa soluzione (f), scritta sotto forma di algoritmo appartenente ad un linguaggio di programmazione, tale che la sua esecuzione dia la soluzione. Più formalmente:

$$\mathbb{N} \rightarrow \mathbb{N} \ni f \quad \exists A \in \mathcal{L} : \llbracket A \rrbracket = f$$

1.3 Teorema di Cantor (1874)

Il seguente teorema ha come conseguenza che **esistono insiemi non numerabili**. Questo risultato si attribuisce a Georg Cantor, matematico tedesco, nel 1874.

La **dimostrazione** è importante da capire. Essa utilizza una tecnica, detta dimostrazione diagonale, che è alla base di gran parte dei risultati principali che stabiliscono i fondamenti dell'informatica come scienza (Dauben, 1979; [Official Cambridge article link](#)).

Teorema 1 (Cantor).

$$|\mathbb{N}| < |\mathbb{N} \rightarrow \mathbb{N}| \quad (1)$$

La cardinalità di \mathbb{N} (numero di programmi per risolvere problemi) è strettamente più piccolo della cardinalità delle funzioni $\mathbb{N} \rightarrow \mathbb{N}$ (numero di problemi esistenti).

Dimostrazione. Si supponga per assurdo che $|\mathbb{N}| = |\mathbb{N} \rightarrow \mathbb{N}|$. Questo implica che esistono funzioni numerabili come per esempio $f_0, f_1, f_2, \dots, f_x, \dots$

La genialità di Cantor si manifesta quando pensa ad una funzione $g(x)$ così definita:

$$g(x) = f_x(x) + 1 \quad \text{con } g : \mathbb{N} \rightarrow \mathbb{N}$$

Con ovviamente $x \in \mathbb{N}$. La funzione $g(x)$ prende un numero naturale e restituisce un numero naturale, quindi è correttamente identificabile come un problema (definizione di problema a pagina 3) e matematicamente formalizzabile come $\mathbb{N} \rightarrow \mathbb{N}$.

Dunque, prendendo qualsiasi funzione f numerata x -esima, essa sarà diversa dalla funzione g numerata x -esima poiché sempre aumentata di 1:

$$f_x(x) \neq g(x) \rightarrow f_x(x) \neq f_x(x) + 1$$

QED

Questo teorema purtroppo non è possibile applicarlo agli algoritmi informatici poiché se al posto della funzione $f_x(x)$ venisse inserito un algoritmo e quest'ultimo non terminasse mai, dunque sostituibile con ∞ , la somma +1 non verrebbe mai eseguita. Per esempio, quando un programma entra in un loop che non gli consente di eseguire le istruzioni successive.

1.4 Problema decisionale e ipotesi del continuo

Un **alfabeto** è una sequenza di simboli con cui è possibile scrivere gli algoritmi risolutivi. L'alfabeto utilizzato nelle realizzazioni tecnologiche è l'**alfabeto binario** $\Sigma = \{0, 1\}$.

Un **problema decisionale** è la versione associata ad un dato problema informatico $f : \mathbb{N} \rightarrow \mathbb{N}$, ovvero alla funzione:

$$d_f : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\} \quad (2)$$

Definita nel seguente modo:

$$d_f((x, y)) = \begin{cases} 1 & \text{se } y = f(x) \\ 0 & \text{altrimenti} \end{cases}$$

Un problema decisionale non è altro che una funzione con co-dominio $\{0, 1\}$ che è in grado di decidere se una data coppia $(x, y) \in \mathbb{N} \times \mathbb{N}$ appartiene ad f .

Essendo un problema decisionale una funzione associata ai problemi in informatica, allora esiste la relazione:

$$\mathbb{N} \rightarrow \{0, 1\} \subseteq \mathbb{N} \rightarrow \mathbb{N}$$

Dunque, sicuramente sarà vera la seguente condizione:

$$|\mathbb{N} \rightarrow \{0, 1\}| \leq |\mathbb{N} \rightarrow \mathbb{N}|$$

Ma sarà vera anche la seguente:

$$|\mathbb{N} \rightarrow \{0, 1\}| = |\mathbb{N} \rightarrow \mathbb{N}|$$

Dimostrazione. È chiaro che la seguente relazione è vera:

$$|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$$

Allora, vale anche:

$$|\mathbb{N} \rightarrow \{0, 1\}| = |\mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}|$$

Si prenda qualsiasi funzione del tipo $f : \mathbb{N} \rightarrow \mathbb{N}$. A tale funzione, viene associato l'insieme:

$$S_f = \{(i, o) \mid f(i) = o\} \subseteq \mathbb{N} \times \mathbb{N}$$

In cui i indica l'input e o indica l'output. Viene scritta la sua relativa equazione caratteristica:

$$f_{S_f}(x, y) = \begin{cases} 1 & \text{se } (x, y) \in S_f \\ 0 & \text{altrimenti} \end{cases}$$

Allora si può affermare con certezza:

$$|\mathbb{N}| < |\mathbb{N} \rightarrow \mathbb{N}| = |\mathbb{N} \rightarrow \{0, 1\}| = |2^{\mathbb{N}}| = |\mathbb{R}| \quad (3)$$

L'ultima uguaglianza è possibile grazie all'**ipotesi del continuo**. QED

2 Linguaggi regolari ed automi a stati finiti

2.1 Alfabeti e Linguaggi

Qui di seguito si lasciano una serie di definizioni utili per il futuro:

- ☛ **Simbolo.** Entità primitiva astratta che non viene definita formalmente (come punto, linea, etc.).
Per esempio, lettere e caratteri numerici sono simboli.
- ☛ **Alfabeto.** Rappresentato con la lettera greca Sigma Σ , è un **insieme finito di simboli**.
- ☛ **Stringa (o parola).** Sequenza finita di simboli giustapposti, ovvero messi uno affianco all'altro.
Per esempio, se a, b, c sono simboli, allora $abcb$ è una stringa.
- ☛ **Lunghezza di una stringa.** Viene denotata con $|w|$, in cui w è una stringa, e rappresenta il **numero di occorrenze di simboli che compongono una stringa**. Ad esempio, $|abcb| = 5$.
Attenzione che la **stringa vuota** viene denotata con ε ed è la **stringa costituita da zero simboli**: $|\varepsilon| = 0$.
- ☛ **Concatenazione di stringhe.** Due stringhe v e w sono concatenate quando si rappresentano nel seguente modo vw . Si **ottiene facendo seguire alla prima stringa la seconda**.
La concatenazione è un'operazione che ammette come identità la stringa vuota ε .
- ☛ **Linguaggio formale.** Detto anche linguaggio L , è un **insieme di stringhe di simboli da un alfabeto Σ** . L'insieme vuoto \emptyset e l'insieme $\{\varepsilon\}$ sono due linguaggi formali di qualunque alfabeto. L'insieme \emptyset **non contiene elementi**, mentre l'insieme $\{\varepsilon\}$ **ne contiene uno**, ovvero la stringa vuota → sono insiemi diversi!
- ☛ **Sequenze finite.** Rappresentate con Σ^* , è il **linguaggio costituito da tutte le stringhe su un fissato alfabeto Σ** . Quindi, viene considerato anche il **linguaggio più grande esistente**, ovvero il limite superiore. Tuttavia, questo non implica che sia il linguaggio più efficiente o potente (argomento approfondito in futuro). In parole povere, è l'insieme delle sequenze di stringhe finite.
Definizione matematica:

$$\Sigma^* = \{a_1 \cdots a_n \mid n \geq 0, a_i \in \Sigma\}$$

Sia quindi $L \subseteq \Sigma^*$ un **generico linguaggio formale** sull'alfabeto Σ .

2.2 Operazioni sui linguaggi

Sia Σ un alfabeto e L, L_1, L_2 insiemi di stringhe di Σ^* . Le **operazioni sui linguaggi** sono principalmente tre:

- ☞ **Concatenazione.** La **concatenazione** di L_1 e L_2 , denotata con $L_1 \cdot L_2$ è l'insieme:

$$L_1 L_2 = \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}$$

Si definisce dunque:

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L \cdot L^n \end{cases}$$

E facendo attenzione al fatto $L^0 = \{\varepsilon\} \neq \emptyset$.

- ☞ **Complemento.** Il **complemento** di un linguaggio è denotato con \bar{L} :

$$\bar{L} = \{\sigma \mid \sigma \in \Sigma^*, \sigma \notin L\}$$

- ☞ **Chiusura di Kleene.** Viene denotata con L^* ed è l'insieme così definito:

$$L^* = \bigcup_{n \geq 0} L^n$$

- ☞ **Chiusura positiva.** Denotata con L^+ è l'insieme così definito:

$$L^+ = \bigcup_{n \geq 1} L^n$$

E si verifica immediatamente che $L^+ = LL^*$. Quindi, questo operato si può derivare dalla chiusura e dalla concatenazione.

- ☞ **Unione.** L'**unione** tra due linguaggi, che corrisponde ad un ***or logico***, è così definita:

$$L_1 \cup L_2 = \{\sigma \mid \sigma \in L_1 \vee \sigma \in L_2\}$$

- ☞ **Intersezione.** L'**intersezione** tra due linguaggi, che corrisponde ad un ***and logico***, è così definita:

$$L_1 \cap L_2 = \{\sigma \mid \sigma \in L_1 \wedge \sigma \in L_2\}$$

Esempio di esercizio su linguaggi e alfabeti

Dati i seguenti dati:

Alfabeto: $\Sigma = \{a, b\}$

Linguaggio: $L = \{a, b\}$

Si costruisce l'insieme L^* , ovvero il linguaggio costituito da tutte le stringhe su un fissato alfabeto. Per definizione uguale anche a Σ^* :

$$L^* = \{\varepsilon, a, b, aa, ab, ba, bb, \dots\} = \Sigma^*$$

In cui la prima lettera, ovvero ε , può essere rappresentato con L^0 ;

Le lettere $\{a\}$ e $\{b\}$, insieme a L^0 , possono essere rappresentate con L^1 ;

Le lettere $\{aa\}$, $\{ab\}$, $\{ba\}$, $\{bb\}$, insieme a L^0 e L^1 , possono essere rappresentate con L^2 ;

E così via. Date che sono infinite stringhe componibili, il linguaggio viene associato a Σ^* .

2.3 Automa a stati finiti

Un **automa a stati finiti** è un modello matematico di un sistema avente un input ed eventualmente un output, a valori discreti. Il sistema può essere in uno stato tra un insieme finito di stati possibili. Essendo in uno stato, l'automa ha la possibilità di **tenere traccia della storia precedente**.

Analizzando letteralmente le parole di “automa a stati finiti”:

- ☞ **Automa.** Macchine che lavorano indipendentemente dall'intervento dell'essere umano.
- ☞ **A stati finiti.** Con **stato** si intende lo stato effettivo della macchina. Mentre con **finiti** si intende che lo stato al tempo t è effettivamente finito, ovvero è una quantità di informazione finita.

Solitamente, viene rappresentato con una testina che legge da un nastro, quest'ultimo contenente la sequenza di simboli dell'alfabeto dati in input all'automa. La testina che legge si sposta sempre nella stessa direzione, consumando la sequenza in input. La testina si può trovare in un certo **stato**; a seconda dello stato q e del simbolo s_i letto, la testina si porta in un certo altro stato (o rimane nello stesso) e si sposta a destra per apprestarsi a leggere il simbolo successivo dalla sequenza.

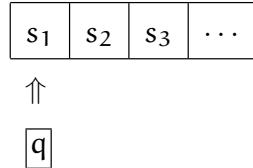


Figura 1: Dispositivo che rappresenta un esempio generalistico di un automa a stati finiti.

Una volta terminata la lettura, l'automa è in grado di fornire il risultato di accettazione o di refutazione della stringa (parola) letta. Il **comportamento** dell'automa si **definisce** in maniera univoca mediante una tabella, chiamata **matrice di transizione**, come ad esempio:

	<i>a</i>	<i>b</i>
<i>q</i> ₀	<i>q</i> ₁	<i>q</i> ₂
<i>q</i> ₁	<i>q</i> ₁	<i>q</i> ₀
<i>q</i> ₂	<i>q</i> ₁	<i>q</i> ₀

Tabella 1: Matrice di transizione.

Tuttavia, la rappresentazione più comune e chiara è il **grafo**:

- Gli **archi** rappresentano le *transizioni* etichettate con il simbolo in lettura;
- Lo **stato iniziale** è rappresentato con la *freccia “start”*;
- Gli **stati finali** sono cerchiati due volte.

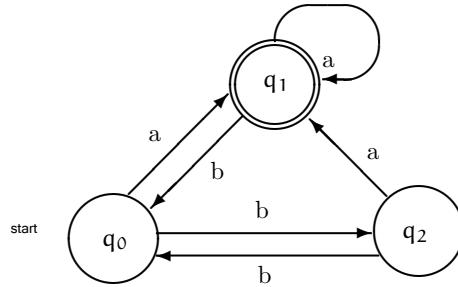


Figura 2: Esempio di grafo (freccia start mancante).

Un **linguaggio vuoto** è il più piccolo dei linguaggi, si rappresenta con il simbolo \emptyset e si riconosce poiché **non ha stati finiti** (quindi nessun nodo con il doppio cerchio). Infine, il **linguaggio più grande** è il linguaggio più grande, si rappresenta con il simbolo Σ^* , esiste dunque la relazione $\emptyset \subseteq \Sigma^*$ e si riconosce perché **ha solo stati finiti** (quindi nessun nodo con un solo cerchio).

2.3.1 Automi deterministici (DFA)

Un **automa a stati finiti deterministico** (DFA) è una che, date determinate condizioni, è possibile determinare la serie di operazioni. Viene rappresentato con una quintupla del tipo $\langle Q, \Sigma, \delta, q_0, F \rangle$ in cui:

- Q è un insieme di **stati finiti**;
- Σ è un **alfabeto finito**, cioè un alfabeto di input;
- $\delta : Q \times \Sigma \rightarrow Q$ è la **funzione di transizione** che dato lo stato $q \in Q$ in cui si trova la macchina ed un simbolo $a \in \Sigma$ in lettura del nastro, produce il prossimo stato $\delta(q, a) \in Q$ in cui si troverà la macchina;
- q_0 è lo **stato iniziale**;
- $F \subseteq Q$ è l'insieme degli **stati finali**, anche detti **di accettazione**.
- $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ si ottiene dalla funzione δ ed è definita nel seguente modo:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = q \\ \hat{\delta}(q, wa) = \delta\left(\hat{\delta}(q, w), a\right) \end{cases}$$

Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti deterministico. Una **stringa** x è detta **accettata** da un M se $\hat{\delta}(q_0, x) \in F$. Inoltre, il **linguaggio** è **accettato dalla macchina** M , denotato come $L(M)$, è l'insieme di tutte le stringhe accettate da M , ovvero:

$$L(M) = \left\{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F \right\}$$

N.B. A lezione al posto di w e x è stato utilizzato σ .

Un linguaggio L viene detto **linguaggio regolare** se è accettato da qualche automa a stati finiti deterministico (DFA), ovvero se esiste M tale che $L = L(M)$.

2.3.2 Esempio esercizio (automati deterministici)

Esercizio.

Il grafo dell'esercizio è il seguente:

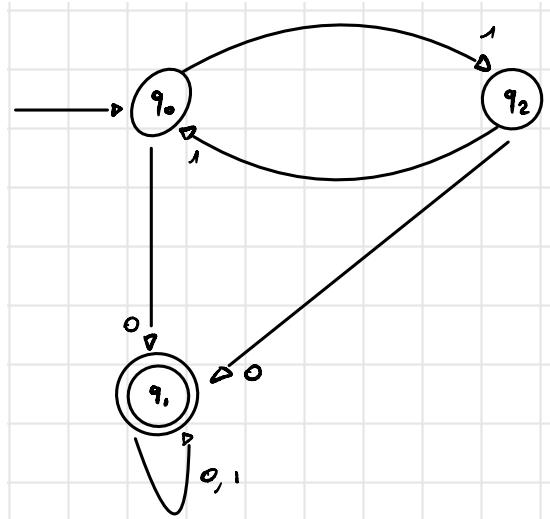


Figura 3: Grafo di un automa a stati finiti deterministico.

I dati forniti sono i seguenti:

Linguaggio: $L(M) = \{x \in \{0, 1\}^* \mid x \text{ contiene almeno uno } 0\}$

Alfabeto: $\Sigma = \{0, 1\}$

Si dimostri che $L(M)$ è \subseteq e \supseteq del linguaggio che è stato dato. Quindi, è necessario verificare le condizioni con:

$$\{x \in \{0, 1\}^* \mid x \text{ contiene almeno uno } 0\}$$

Risoluzione.

La stringa x è sicuramente formata da:

$$x = 1^n 0 w \in \{0, 1\}^* \text{ e } n \in \mathbb{N}$$

In cui ci deve essere un numero n di concatenazioni di numeri 1, uno 0 come viene specificato “ x contiene almeno uno 0” e infine una qualsiasi stringa dell’alfabeto w . L’**obiettivo** è dimostrare che:

$$\forall w \in \{0, 1\}^* : \delta(q_1, w) = q_1$$

Ovvero che dato qualsiasi simbolo appartenente all’alfabeto, lo stato successivo sia sempre q_1 , ovvero quello finale!

Si supponga che la cardinalità di w , ovvero la lunghezza della stringa, sia uguale a m , cioè $|w| = m$. Questo implica che lo stato non varia:

$$m = 0 \implies w = \varepsilon \quad \text{e quindi} \quad \hat{\delta}(q_1, w) = q_1$$

Invece, il caso in cui la stringa non sia vuota:

$$m \geq 0 \text{ e } m + 1 \implies w = \sigma 0 \quad \text{oppure} \quad w = \sigma 1$$

Per **induzione** si riesce a dimostrare che:

$$\hat{\delta}(q_1, \sigma 0) = \delta\left(\hat{\delta}(q_1, \sigma), 0\right) = \delta(q_1, 0) = q_1$$

Grazie all'induzione è stato dimostrato che se era vero per m , allora è vero anche per $m + 1$, quindi qualsiasi m (maggiore o uguale a zero ovviamente). Quindi, il **primo passo dell'esercizio è stato concluso, ovvero quello di dimostrare che la w è sempre accettata soltanto se posta dopo il primo zero** (sequenza conclusiva dell'automa a stati finiti deterministico).

Adesso si prosegue la dimostrazione dimostrando che per ogni numero naturale, con una sequenza di 1 si finisce nello stato q_0 o q_2 , ovvero l'automa non termina.

Caso base:

- $n = 0 : \hat{\delta}(q_0, \varepsilon) = q_0 \in \{q_0, q_2\}$
- $n \geq 0 : \hat{\delta}(q_0, 1^{n+1}) = \hat{\delta}(q_0, 1^n 1) = \delta\left(\hat{\delta}(q_0, 1^n)\right) \in \{q_0, q_2\}$

✓**Dimostrato** che ogni stringa in forma $x \in L$ (ogni stringa nel linguaggio), è accettata dall'automa. Ovvero $L \subseteq L(M)$.

L'esercizio si conclude con la dimostrazione $L \supseteq L(M)$ - Quindi se x non contiene almeno uno zero ($x = 1^n$), allora si può affermare con certezza che x non è in $L(M) = \left\{ \sigma \mid \hat{\delta}(q_0, \sigma) = q_1 \right\}$. Quindi formalmente:

$$\forall n : 1^n \implies \hat{\delta}(q_0, x) \in \{q_0, q_2\} \quad \checkmark \text{Dimostrato}$$

2.3.3 Automi non-deterministici (NFA)

Un **automa a stati finiti non-deterministico** (NFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove Q, Σ, q_0 e $F \subseteq Q$ mantengono il significato visto per gli automi deterministici (pagina 11), mentre la **funzione di transizione** δ è definita come:

$$\delta : Q \times \Sigma \longrightarrow P(Q) = 2^{|Q|}$$

Ovvero è una relazione tra stati.

In particolare, si tratta di un concetto fondamentale in informatica che definisce un **modello (ideale)** di calcolo parallelo su cui si fonda la moderna analisi della complessità degli algoritmi.

Adesso è possibile avere $\delta(q, a) = \emptyset$ per qualche $q \in Q$ ed $a \in \Sigma$, poiché l'**automa non può avere transizioni per alcuni simboli in input**.

Si definisce la funzione $\hat{\delta} : Q \times \Sigma^* \longrightarrow P(Q)$ nel seguente modo:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = \{q\} \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \delta(p, a) \end{cases}$$

Inoltre, si dice che **una stringa x è accettata da un automa a stati finiti non-deterministici NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$** se:

$$\hat{\delta}(q_0, x) \cap F \neq \emptyset$$

In altre parole, una stringa è accettata quando una di queste computazioni raggiunge uno stato finale dopo aver consumato la sequenza in input.

Invece, si dice che **un linguaggio è accettato da un automa a stati finiti non-deterministici NFA M** se corrisponde all'insieme delle stringhe accettate:

$$L(M) = \left\{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset \right\}$$

La rappresentazione a grafo rimane pressoché immutata. L'**unica differenza** è che da un nodo possono uscire più archi (o nessuno) etichettati dallo stesso simbolo.

2.3.4 Teorema Rabin-Scott (1959)

Teorema 2 (Rabin-Scott). *Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un automa a stati finiti non deterministico (NFA). Allora esiste un automa a stati finiti deterministico M' tale che $L(M) = L(M')$.*

Dimostrazione. Si definisce l'automa a stati finiti non deterministico con la quintupla $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$ e con le seguenti proprietà:

$$\star \quad \Sigma' = \Sigma.$$

$\star \quad Q' = P(Q)$, sarebbe più preciso definire $Q' = \{q_1, \dots, q_{2|Q|}\}$ e poi stabilire una corrispondenza biunivoca fra tali stati e gli elementi di $P(Q)$. Tuttavia, così facendo rimane più chiara la dimostrazione.

$$\star \quad q'_0 = \{q_0\}.$$

$$\star \quad F' = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$$

$$\star \quad \delta'(P, a) = \bigcup_{p \in P} \delta(p, a), \text{ per ogni } P \in P(Q)$$

Si mostra, **per induzione**, sulla lunghezza della stringa di input $\sigma \in \Sigma^*$ che:

$$\forall \sigma \in \Sigma^* : \hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x) \quad (4)$$

In cui la parte di sinistra è per gli automi non deterministici ($\hat{\delta}(q_0, x)$), mentre la parte di destra è per gli automi deterministici ($\hat{\delta}'(q'_0, x)$).

Caso base.¹

$$|\sigma| = 0 \iff \sigma = \varepsilon : \hat{\delta}(q_0, \varepsilon) = \{q_0\} = q'_0 = \hat{\delta}'(q'_0, \varepsilon)$$

Passo induttivo.

$$\forall \sigma \in \Sigma^* : |\sigma| \leq n : \hat{\delta}(q_0, \sigma) = \hat{\delta}(q_0, \sigma)$$

Quindi che non ci siano gli apici ' come nell'equazione 4. Si dimostra induttivamente:

$$\hat{\delta}'(q'_0, \sigma a) = \delta'\left(\hat{\delta}'(q'_0, a), a\right) = \delta'\left(\hat{\delta}(q_0, \sigma), a\right) = \bigcup_{p \in \hat{\delta}(q_0, \sigma)} \delta(p, a) = \hat{\delta}(q_0, \sigma a)$$

In cui $p \in \hat{\delta}(q_0, \sigma)$ e $\hat{\delta}(q_0, \sigma a)$ sono **macchine deterministiche**.

La dimostrazione si conclude con le seguenti ovvie relazioni:

$$\sigma \in L \iff \hat{\delta}(q_0, \sigma) \cap F \neq \emptyset \iff \hat{\delta}'(q'_0, \sigma) \cap F \neq \emptyset \iff \hat{\delta}'(q'_0, \sigma) \in F' \iff \sigma \in L(M')$$

QED

¹Il simbolo \iff indica "se e solo se".

2.3.5 Esempio esercizio (automati non-deterministici)

Esercizio.

Il grafo dell'esercizio è il seguente:

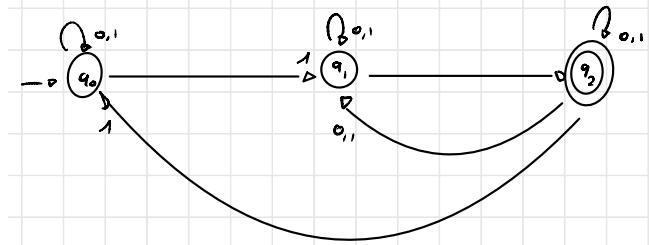


Figura 4: Grafo di un automa a stati finiti non-deterministico.

Risoluzione.

La Q è formata da $\{q_0, q_1, q_2\}$, mentre la $P(a)$ ha i seguenti insiemi:

$$P(a) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

Le espressioni in rosso sono sequenze finite che terminano.

Infine, l'automa non-deterministico è riconducibile a un automa deterministico:

$$\delta'(s, a) = \bigcup_{q \in S} \delta(q, a)$$

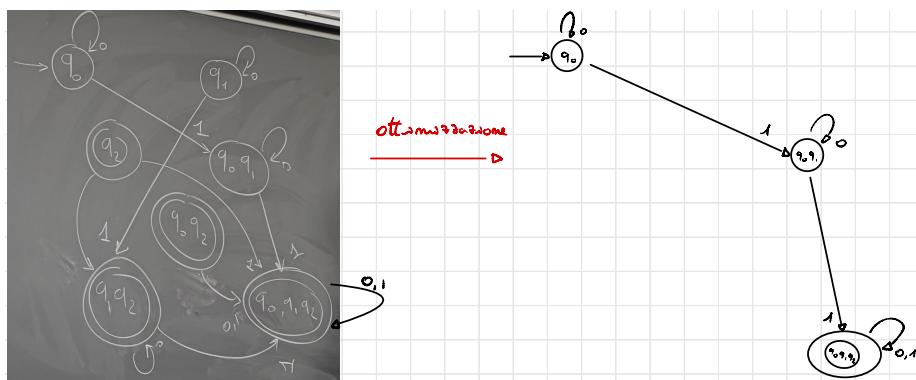


Figura 5: Rappresentazione di una conversione da non-deterministico a deterministico con relativa ottimizzazione.