Fondamenti di informatica Madonia

Linguaggi Formali

Un **linguaggio formale** è un insieme di parole costruite a partire da un **alfabeto** finito.

Esempio:

Nel DNA, le 4 basi azotate (A, T, C, G) possono essere viste come un alfabeto. Le sequenze di DNA sono parole costruite con queste basi, seguendo determinate regole.

A differenza dei linguaggi naturali (es. italiano), i linguaggi formali seguono regole rigorose e possono essere studiati matematicamente.

Relazione con la Teoria dell'Informazione

La **teoria dell'informazione** ha influenzato lo sviluppo dei linguaggi formali. Questa teoria, basata sulla codifica binaria (0 e 1), ha permesso di analizzare la trasmissione e l'elaborazione delle informazioni.

I linguaggi formali sono anche strettamente legati all'algebra, in particolare a strutture come **anelli**, **gruppi** e **campi** (strutture algebriche).

Alfabeto e Parole

Un **alfabeto** Σ è un insieme finito di simboli.

Esempi di alfabeti:

- Σ = {0,1} (alfabeto binario)
- Σ = {A, B, C}
- Σ = {a, b, ..., z} (alfabeto latino)

Una **parola** o **stringa** è una sequenza finita di simboli di un alfabeto.

Esempio:

- 001 è una parola sull'alfabeto Σ = {0,1}.
- "abc" è una parola sull'alfabeto Σ = {a, b, ..., z}.

Le **parole infinite** sono chiamate *omega words* e non vengono trattate nei linguaggi finiti.

La **parola vuota** ε (epsilon) è la parola che non contiene alcun simbolo.

L'insieme di tutte le possibili parole su un alfabeto Σ si indica con **Σ*** (sigma star).

N.B. Parliamo sempre di parole di lunghezza finita

N.B. Il numero di parole di un alfabeto finito è infinito.

Definizione di Σ*

Formalmente, dato un alfabeto Σ:

1. La parola vuota ε ∈ Σ*.
2. Se x ∈ Σ* e a ∈ Σ, allora anche xa ∈ Σ*.

Esempio:

Dato Σ = {0,1}, la parola "010" appartiene a Σ*?

Sì, perché posso considerare 01 come una x che viene seguito da a che sarebbe lo 0. Per la seconda definizione allora esso appartiene a Σ*.

Per vedere se un singolo simbolo appartiene a Σ* possiamo allora seguire sempre la nostra seconda definizione mettendo che ε appartenga a Σ* (come la nostra definizione 1) e poi posso considerare il singolo simbolo come a ∈ Σ.

Concatenazione di parole

Se x e y sono due parole in Σ*, la loro **concatenazione** si indica con "·" e rappresenta la sequenza di x seguita da y.

Proprietà della concatenazione:

1. **Operazione interna**: la concatenazione di due parole appartiene ancora a Σ*.
2. **Elemento neutro**: la parola vuota ε è l'elemento neutro.
3. **Associatività**: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

La concatenazione **non è commutativa** in generale.

Monoide Sintattico: L'insieme Σ* con l'operazione di concatenazione e l'elemento neutro ε forma una struttura chiamata **monoide**.

Potenza di una stringa

Sia Σ un alfabeto e sia $n \in \Sigma^*$. La potenza n-esima di x, con $x \geq 0$, n indice con x^n ed è così definita:

1. per n = 0 $x^0 = \varepsilon$
2. per n > 0 $x^n = x^{n-1} \cdot x$

Linguaggi Formali

Un **linguaggio formale** L è un sottoinsieme di Σ*.

Definizione: Ogni L ⊆ Σ* è un linguaggio.

Esempi:

- **Linguaggio finito**: L = {00, 01, 10, 11} su Σ = {0,1}, contiene solo le parole di lunghezza 2.
- **Linguaggio infinito**: L = { ax | a = 0 e $x \in \Sigma^*$ }, ovvero tutte le parole su Σ che iniziano con 0.

Linguaggio Vuoto e Parola Vuota

- L'insieme vuoto ∅ è un linguaggio (senza parole) ed è indicato con λ.
- L'insieme { ε } è un linguaggio contenente solo la parola vuota. (perché dobbiamo verificare che ogni elemento di quel linguaggio appartiene a Σ*. Essendo ε ∈ Σ^* allora è un linguaggio).

Σ^n sarebbe l'insieme di tutte le stringhe che hanno lunghezza esattamente n.

La lunghezza di una stringa si indica con $|x|$ cioè la lunghezza della stringa x.

N.B. Non confondere la lunghezza di una stringa con la cardinalità di un insieme.

Corrispondenza Biunivoca

Possiamo stabilire una **corrispondenza biunivoca** tra le stringhe che iniziano con "0" e l'insieme Σ*. Basta rimuovere il primo carattere da ogni stringa di L per ottenere un elemento di Σ*. Poiché Σ* è infinito, anche il linguaggio L è infinito.

Definizione di Linguaggi

Quando un linguaggio è finito, lo possiamo elencare esplicitamente con le parentesi graffe { }. Se è infinito, dobbiamo descriverlo usando **strumenti finiti**, come espressioni regolari o grammatiche formali.

Metodi per Definire un Linguaggio

1. **Metodi riconoscitivi**: si fornisce un input x e si ottiene una risposta "sì" o "no" in base all'appartenenza al linguaggio.
2. **Metodi generativi**: si definiscono regole per costruire tutte le stringhe del linguaggio (es. grammatiche formali).

Esiste una **corrispondenza tra metodi riconoscitivi e generativi**, cioè ogni linguaggio definibile con uno dei due metodi può essere espresso anche con l'altro.

Concatenazione di Linguaggi

Dati due linguaggi L_1 e L_2, la loro concatenazione è: $L_1 \cdot L_2 = \{ xy \mid x \in L_1 \text{ e } y \in L_2 \}$.

Proprietà della concatenazione:

- Se almeno uno dei linguaggi è infinito, il risultato è infinito.
- Non è commutativa ($L_1 \cdot L_2 \neq L_2 \cdot L_1$).
- Se $L_1 = \emptyset$, allora $L_1 \cdot L_2 = \emptyset$.
- Se $L_1 = \{ \varepsilon \}$, allora $L_1 \cdot L_2 = L_2$.

Potenza di un Linguaggio

Definiamo la potenza n-esima di un linguaggio come:

1. Se $n > 0$ $\quad L^n = L \cdot L^{n-1}$
2. Se $n = 0$ $\quad L^0 = \{ \varepsilon \}$.

Concatenazione tra L1 e L2

Iniziamo con un quesito: L1 è sempre un sottoinsieme di L1·L2? Per capire meglio la concatenazione tra due linguaggi, consideriamo il numero di elementi che ne risultano. Se entrambi i linguaggi, L1 e L2, sono finiti, anche il risultato della concatenazione sarà finito. Al contrario, se almeno uno dei due linguaggi è infinito, anche la concatenazione produrrà un linguaggio infinito.

Quando L1 e L2 sono entrambi finiti, possiamo chiederci quanti elementi conterrà la loro concatenazione. Se indichiamo con $|L1|=h$ e $|L2|=k$ le cardinalità dei due linguaggi, allora la cardinalità della concatenazione sarà data dal prodotto:

$$|L1| \times |L2|$$

Es. se L1={aa,aab} e L2={bb,b}, allora la concatenazione L1·L2 sarà {aabb,aab,aabbb,aabb}. In generale, possiamo affermare che la cardinalità della concatenazione di due linguaggi è minore o uguale al prodotto delle loro cardinalità:

$$|L1 \cdot L2| \leq |L1| \times |L2|$$

Potenza di un linguaggio

Passiamo ora al concetto di potenza di un linguaggio. Dato un linguaggio $L \subseteq \Sigma^*$, la sua potenza n-esima, indicata con L^n, è definita in questo modo:

- Se n=0, allora $L^0=\{\epsilon\}$, dove ϵ rappresenta la stringa vuota.
- Se n>0, allora $L^n=L^{n-1} \cdot L$.

Es. $L = \{ab, aa\}$

$L^3 = L^{3-1} \cdot L = L^2 \cdot L$

$L^2 = L^{2-1} \cdot L = L^1 \cdot L$

$L^1 = L^{1-1} \cdot L = L^0 \cdot L = \{\varepsilon\} \cdot L = L$

Es. $L^2 = \{abab, bbbb, abbb, bbab\}$

Chiusura riflessiva di un linguaggio L

Esaminiamo ora il concetto di chiusura di un linguaggio. Iniziamo con la chiusura riflessiva di un linguaggio L, anche detta chiusura di Kleene, indicata con L*. Questa è definita come l'unione di tutte le potenze di L:

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

In altre parole, L* rappresenta tutto ciò che si può ottenere concatenando elementi di L. Possiamo riscrivere questa formula come $L_0 \cup L_1 \cup L_2 \cup L_3 \; ecc$ ...

Dom. La parola vuota appartiene sempre a L*, poiché L_0 contiene la stringa vuota.

Oss. La chiusura è detta riflessiva perché il risultato della concatenazione appartiene ancora ad L*.

Chiusura di L

Consideriamo anche la chiusura di L, o chiusura positiva di L, indicata con L^+. Questa è definita come l'unione delle potenze di L a partire da 1:

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

In questo caso, avremo $L_1 \cup L_2 \cup L_3 \; ecc$ ...

Oss. Se la stringa vuota ϵ appartiene a L, allora appartiene anche a L^+.

Dom. Possiamo scrivere che L* = { ε } ∪ L^+? Sì, perché $L_0 \cup L_1 \cup L_2 \cup L_3 \; ecc$ ... è proprio la definizione di L*.

Dom. Inoltre, possiamo scrivere che L^+ =L*\{ ε }?

- Se $\varepsilon \notin L$, allora possiamo scriverlo.
- Ma se $\varepsilon \in L$, allora non posso scriverlo perché se fosse presente in L lo avrei anche il L^+ ma non in L* perché la toglierei con la formula.

Oss. Un'osservazione interessante è che Σ* è la stessa definizione di L* ma con il linguaggio Σ. Infatti, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$ *ecc* ..., che equivale a $\{\varepsilon\} \cup \Sigma \cup \Sigma \cdot \Sigma \cup \Sigma \cdot \Sigma \cdot \Sigma$ *ecc* ... .Quindi, la risposta alla domanda è sì e viceversa.

Possiamo definire L^* come:

$$\varepsilon \in L^*, \forall x \in L^*, y \in L^* \qquad xy \in l^*$$

Mentre L^+ può essere definito come:

$$\forall x \in L^*, y \in L, \qquad xy \in L^+$$

Es. Ad esempio, se L = {aa} allora L*={$(aa)^n$ | n ≥ 0} = {a^{2n} | n ≥ 0}, ovvero stringhe di lunghezza pari concatenate con sé stesso n volte. Usiamo 2n per indicare che le stringhe devono essere di lunghezza pari.

Metodi di rappresentazione dei linguaggi

I linguaggi possono essere rappresentati attraverso metodi riconoscitori e metodi generativi. Le rappresentazioni finite ci permettono di rappresentare un numero infinito di parole, ovvero linguaggi che contengono un numero infinito di parole.

- Se un linguaggio ha un numero finito di elementi, possiamo rappresentarlo elencando tutti gli elementi tra parentesi graffe {}.
- Tuttavia, se il linguaggio è infinito, dobbiamo utilizzare un metodo riconoscitore o generativo.

I **metodi riconoscitori** sono metodi finiti che prendono una stringa in input e restituiscono una risposta binaria: "sì, la stringa appartiene al linguaggio" oppure "no, la stringa non appartiene al linguaggio".

I **metodi generativi**, come le grammatiche, generano tutti gli elementi di un linguaggio attraverso delle regole. Esistono diversi tipi di grammatiche, tra cui le grammatiche regolari, le grammatiche context-free e le grammatiche senza restrizioni. Gli automi a stati finiti sono equivalenti alle grammatiche regolari.

È utile rappresentare uno stesso concetto con diversi punti di vista e utilizzando diversi strumenti, perché alcuni strumenti possono facilitare la dimostrazione di una proprietà che vale per tutte le altre classi di metodi.

La classe dei linguaggi regolari può essere rappresentata con formule della Logica (Barbanera).

Automi a stati finiti (riconoscitori)

Gli automi a stati finiti sono dei riconoscitori, ovvero strumenti che prendono una stringa in input e restituiscono una risposta binaria.

Un **riconoscitore** è costituito da un **nastro diviso in celle**, dove ogni cella contiene un simbolo alla volta. Utilizziamo i riconoscitori per riconoscere i linguaggi. Sul nastro scriviamo stringhe appartenenti all'alfabeto sigma. Alcune celle del nastro sono vuote e contengono un simbolo speciale chiamato **blank** (b tagliata). Nel modello generale, il nastro è infinito in entrambe le direzioni, ma esistono anche nastri semi-finiti ovvero infiniti solo a destra o solo a sinistra.

I riconoscitori sono **dispositivi teorici**, poiché essendo infiniti non possono essere realizzati praticamente. Il nastro è il luogo in cui si lavora, si scrive e si eseguono i calcoli.

Dopo il nastro c'è una **testina** di lettura e/o scrittura. In alcuni modelli, la testina può solo leggere, mentre in altri può sia leggere che scrivere. La testina legge un simbolo alla volta e può muoversi di una cella alla volta. In alcuni modelli, la testina può spostarsi solo a destra o solo a sinistra, mentre in altri può spostarsi in entrambe le direzioni.

Dopo la testina c'è un **controllo** che si trova in un numero finito di **stati**, che rappresentano le diverse situazioni in cui si trova il riconoscitore.

Il riconoscitore è dotato anche di una **funzione di transizione**, che definisce le regole di comportamento del riconoscitore. Ad esempio, una regola potrebbe essere: "Se sono in questa cella con il simbolo X, allora eseguo l'azione Y".

Quando si lavora con un riconoscitore, si parte da una situazione iniziale e si procede passo dopo passo fino a raggiungere un risultato finale. Ad esempio, in una somma, si legge un simbolo, ci si trova in uno stato, si sostituisce eventualmente il simbolo, ci si sposta per esaminare l'altra parte, si legge un altro simbolo, e così via.

Il movimento del riconoscitore è determinato dalla funzione di transizione, che si comporta in base al simbolo letto nella cella. Il movimento dipende quindi dallo stato corrente e dal simbolo letto.

Questo è il **modello generale di un riconoscitore**.

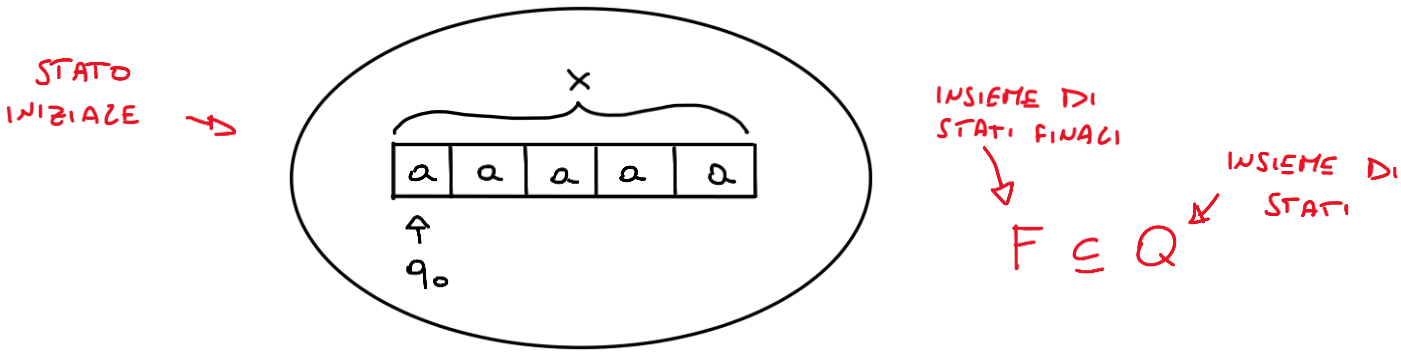
Per utilizzare un riconoscitore, si fornisce una stringa x in input e si avvia il procedimento. La testina inizia a leggere il primo simbolo a partire da uno stato iniziale q_0. Il riconoscitore parte sulla stringa x a partire dalla prima cella, con la testina che legge il primo simbolo e si trova nello stato q_0.

Stato iniziale e stati finali

Se il riconoscitore si trova in uno stato appartenente all'insieme degli **stati finali F**, allora la stringa viene riconosciuta; altrimenti, non viene riconosciuta.

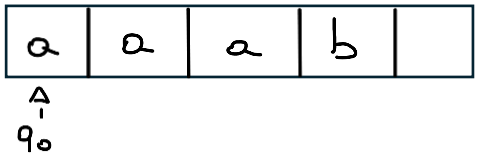
Es. Supponiamo di voler verificare se una stringa ha un numero pari di 'a'. In questo caso, avremo bisogno di tre stati diversi: q_0, $q_{dispari}$, q_{pari}. Inizialmente, leggendo la prima 'a', ci troveremmo nello stato $q_{dispari}$. Lo stato finale del riconoscitore indicherà il risultato.

L'insieme di tutti gli stati possibili è $Q=\{q_0,q_{dispari},q_{pari}\}$, mentre l'unico stato finale accettabile è $F=\{q_{pari}\}$.

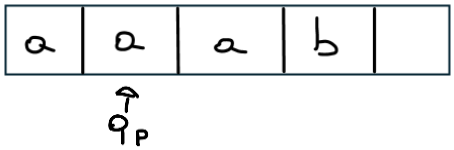


Configurazioni

La situazione iniziale della computazione, ovvero lo stato q_0, è chiamata **configurazione iniziale**. Ovvero:



Una configurazione rappresenta una "fotografia" del riconoscitore in un determinato istante della procedura, fornendo tutte le informazioni sul riconoscitore in quello stato. Da questa configurazione iniziale si passa a una nuova configurazione, chiamata **configurazione successiva**. Ovvero:



La computazione di un riconoscitore è una successione di configurazioni che termina con una configurazione di accettazione o di rifiuto. Il passaggio da una configurazione all'altra è determinato dalle **regole del riconoscitore**.

Indichiamo il passaggio dalla configurazione C_1 alla configurazione C_j sull'automa A con la notazione:

$$C_1 \vdash_A C_j$$

Tra le varie configurazioni raggiungibili da un riconoscitore, abbiamo alcune **configurazioni "particolari"**:

- la configurazione iniziale
- la configurazione di accettazione
- la configurazione di non accettazione.

Le regole per determinare se una configurazione è di accettazione o di rifiuto dipendono dal tipo di riconoscitore. In generale, una configurazione è di accettazione se lo stato corrente q appartiene all'insieme degli stati finali F.

Tano mi ha detto che sono bello <3

In generale, se abbiamo una sequenza di configurazioni finita e massimale (ovvero una configurazione che non ammette configurazioni successive) e se la configurazione finale è di accettazione, allora il riconoscitore accetta la stringa. Questo processo può essere rappresentato come:

$$C_0 \vdash_A C_1 \vdash_A \cdots \vdash_A C_m \quad X$$

Il modello descritto è deterministico. Esistono anche riconoscitori con più nastri, ognuno con la propria testina, ma il funzionamento di base è lo stesso.

Oss. È importante notare che gli stati in un riconoscitore non possono essere infiniti, perché abbiamo bisogno di rappresentare linguaggi infiniti con strumenti finiti. Inoltre, ci occupiamo solo di stringhe di lunghezza finita.

Oss. Anche se il nastro può essere infinito, le stringhe che vi si trovano hanno sempre lunghezza finita. Il nastro infinito serve per non avere limiti di scrittura, ma non implica che si lavora con stringhe infinite. Infatti, in ogni momento della computazione, solo un numero finito di celle del nastro contiene caratteri diversi dal blank.

Macchina di Turing

Lo stato della macchina di Turing può essere paragonato allo stato mentale di una persona che esegue un calcolo, ad esempio, se deve tenere a mente un resto in una somma.

L'idea di Turing era di astrarre il modo in cui la mente umana esegue un calcolo.

Le macchine di Turing possono essere utilizzate non solo per riconoscere linguaggi, ma anche per calcolare funzioni. Se esiste un algoritmo per calcolare una funzione, allora esiste anche una macchina di Turing che può calcolarla. In altre parole, tutte le funzioni calcolabili sono anche Turing-calcolabili.

Un problema fondamentale è l'indecidibilità della macchina di Turing: non esiste un algoritmo che, data in input una macchina di Turing e una stringa, determini se la macchina di Turing termina o meno su quella stringa. Non esiste un algoritmo che possa sistemare l'output della macchina di Turing. L'output della macchina di Turing è indecidibile, e ciò è stato dimostrato partendo dal problema dell'insoddisfacibilità delle formule logiche.

Congettura di Church

La tesi o congettura di Church-Turing afferma infatti che, se esiste un algoritmo per eseguire un compito che manipola simboli, allora esiste una macchina di Turing in grado di eseguire quel compito.

Automa a stati finiti deterministici

Un automa a stati finiti deterministico (ASFD) è un tipo di riconoscitore in cui la testina può solo leggere sul nastro (non scrivere), può spostarsi solo a destra e deve spostarsi ad ogni configurazione successiva.

Un ASFD è definito da una quintupla A=< $\Sigma, Q, q_0, F, \delta >$ dove:

- Σ è l'alfabeto di input.
- Q è l'insieme degli stati (finito).
- $q_0 \in Q$ è lo stato iniziale.
- F è l'insieme degli stati finali.
- δ è la funzione di transizione, che determina il comportamento dell'automa. Questa funzione prende in input lo stato e il simbolo, cioè una coppia {stato, simbolo} e mi restituisce in output uno stato. Esso parte da $Q\ x\ \Sigma$ e dà valore in Q.

$$\delta: Q\ x\ \Sigma \rightarrow Q$$

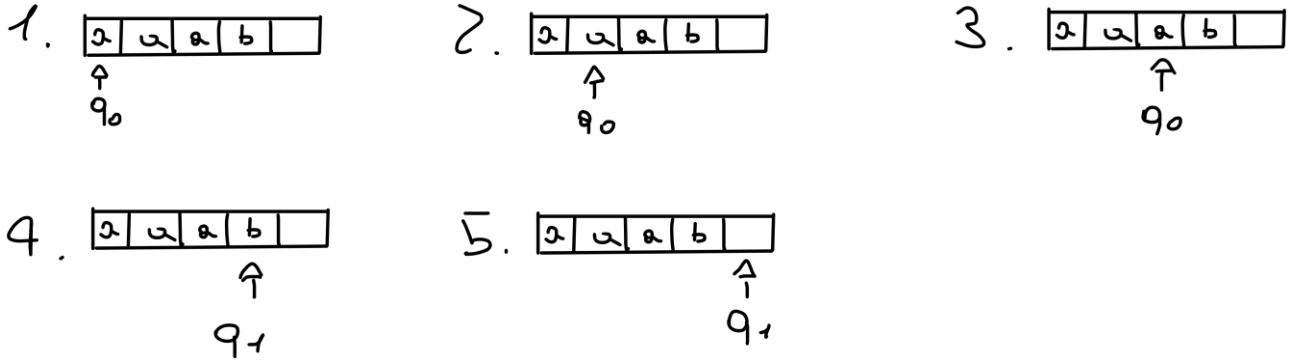
Es. Ad esempio, consideriamo un automa con:

- Q = {q_0, q_1, q_2}
- Σ = {a,b}
- F = {q_1}

- Le seguenti transizioni:
  - δ (q_0, a) = q_0
  - δ (q_1, b) = q_1
  - δ (q_1, a) = q_2
  - δ (q_1, b) = q_2
  - δ (q_2, a) = q_2
  - δ (q_2, b) = q_2

Possiamo rappresentare la funzione di transizione anche con una tabella:

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2



Quando finisco la stringa ci sarà blank e non essendo definito la mia computazione finirà e si avrà il risultato.

Questo automa riconosce le stringhe che, dopo una sequenza di 'a', terminano con una 'b', poiché lo stato finale è q_1.

Esempio di funzionamento dell'automa

Consideriamo la stringa di input "aaab".

1. Inizialmente, l'automa si trova nello stato q0 e la testina punta al primo 'a'.
2. Legge 'a' e si sposta a destra. Lo stato rimane q0 perché $\delta(q0,a)=q0$.
3. Legge 'a' e si sposta a destra. Lo stato rimane q0.
4. Legge 'a' e si sposta a destra. Lo stato rimane q0.
5. Legge 'b' e si sposta a destra. Lo stato diventa q1 perché $\delta(q0,b)=q1$.

Quando la stringa termina, l'automa si trova nello stato q1, che è uno stato finale. Pertanto, l'automa riconosce la stringa "aaab".