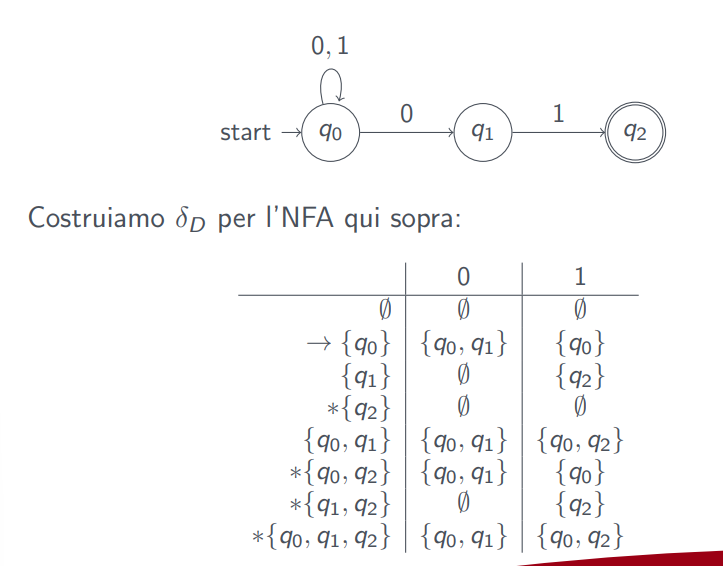
*DFA/NFA*:

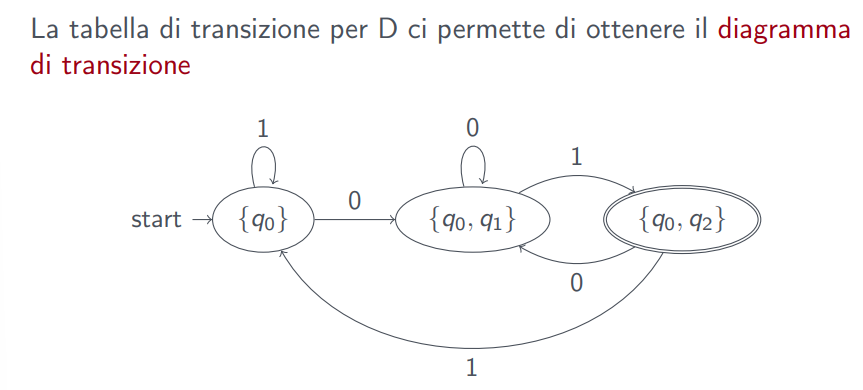
Immagine che contiene testo

Descrizione generata automaticamente

Essi sono in grado di riconoscere lo stesso linguaggio per mezzo della *costruzione per sottoinsiemi*.

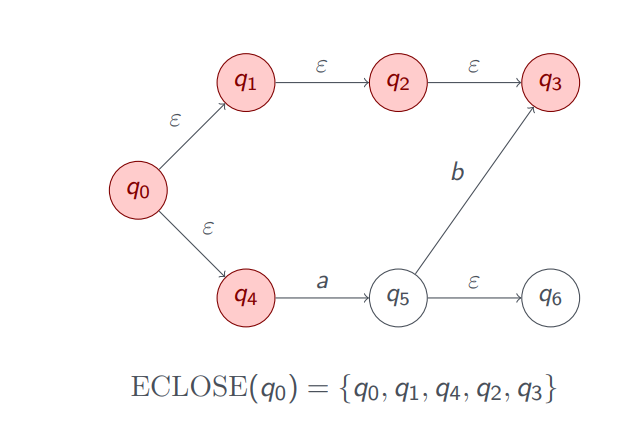
Si costuisce la tabella di transizione per i soli stati raggiunti dallo stato iniziale e si opera correttamente la conversione.





Similmente possiamo avere gli *automi con ε-transizioni*.

In essi si va a calcolare induttivamente la ε-chiusura (tutti gli stati con ε-transizioni raggiunti anche dallo stato iniziale), applicando anche la costruzione a sottoinsiemi per operare la normale conversione da ε-NFA a DFA.



E poi esempio di *conversione DFA a ε-NFA*:

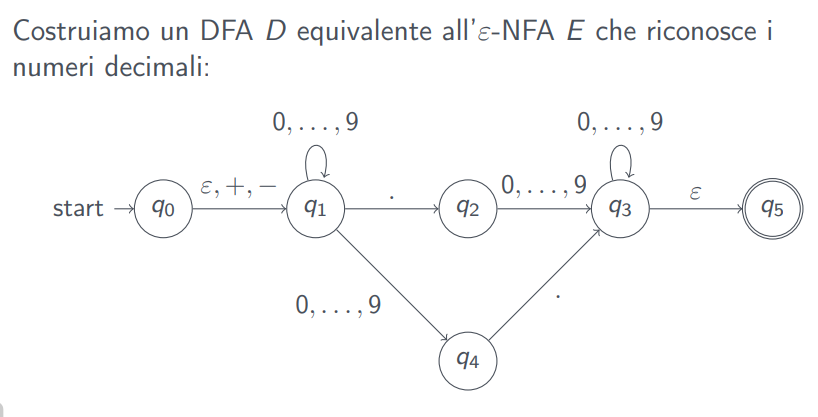
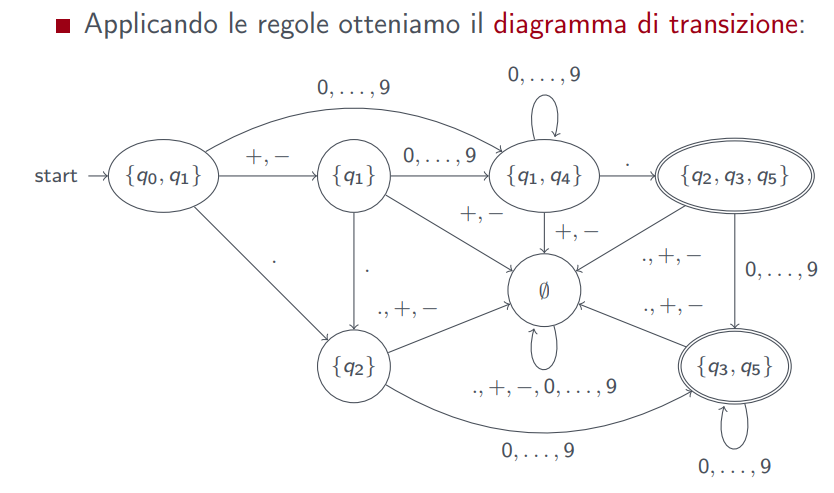


Immagine che contiene testo

Descrizione generata automaticamente



oppure:

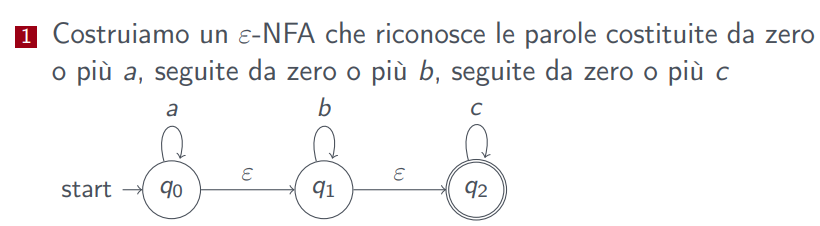
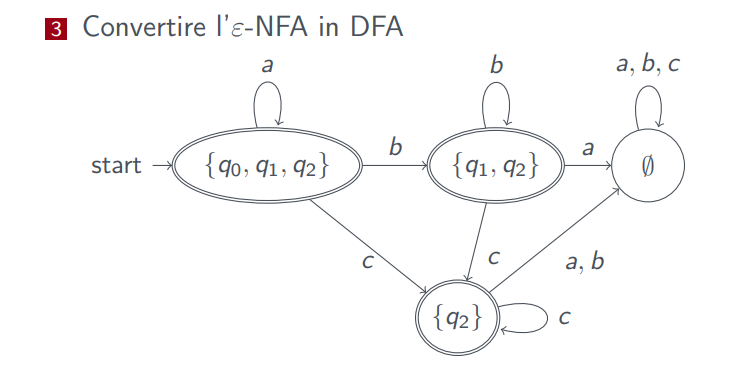


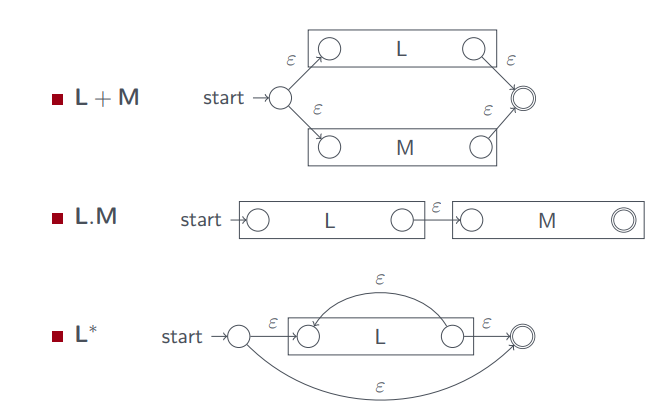
Immagine che contiene testo

Descrizione generata automaticamente



Descriviamo poi le operazioni chiuse degli automi.

Seguono *unione, concatenazione, chiusura di Kleene.*



Un automa riconosce linguaggi regolari; essi vengono scritti tramite *espressioni regolari*.

Immagine che contiene testo

Descrizione generata automaticamente

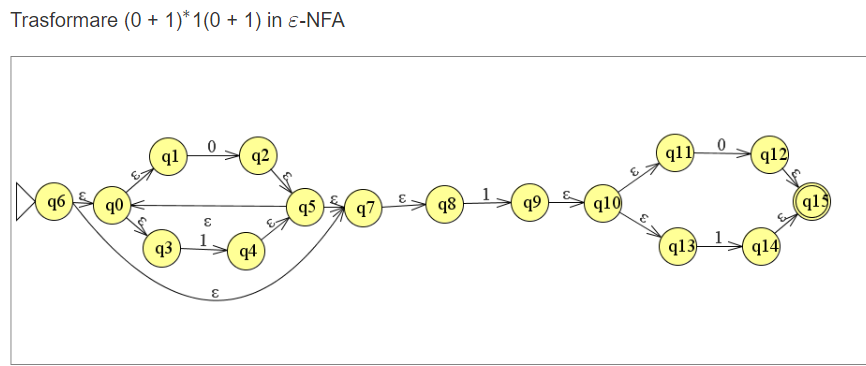
Con un ordine di applicazione:

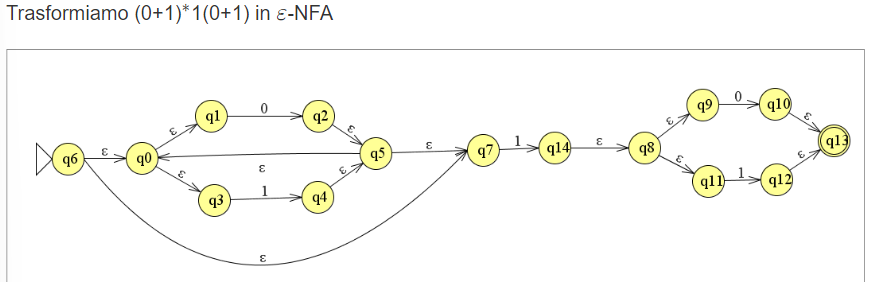
Immagine che contiene testo

Descrizione generata automaticamente

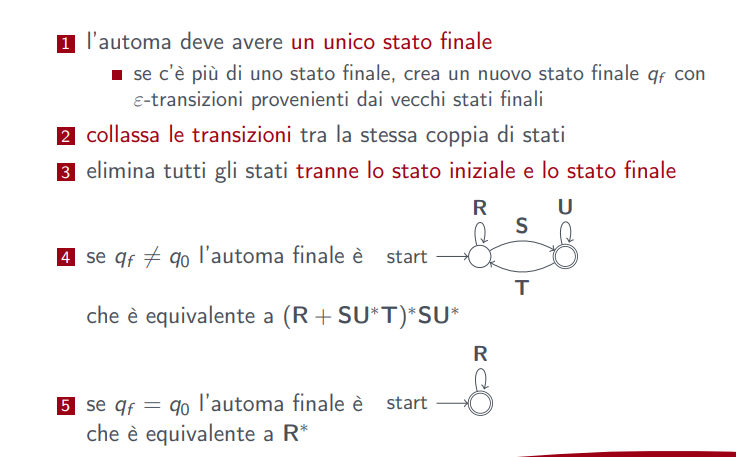
Possiamo anche operare *da RE a ε-NFA* partendo dagli automi sopra descritti.

Esempi pratici:





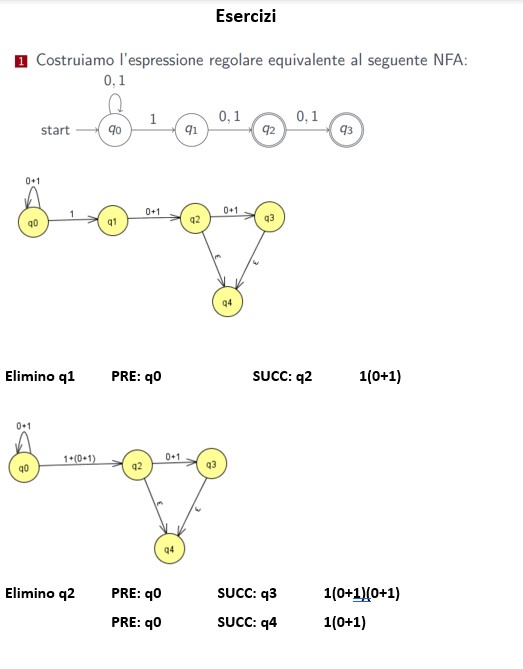
Similmente esiste la *conversione da FA ad RE*, ottenibile per mezzo della *eliminazione degli stati.*

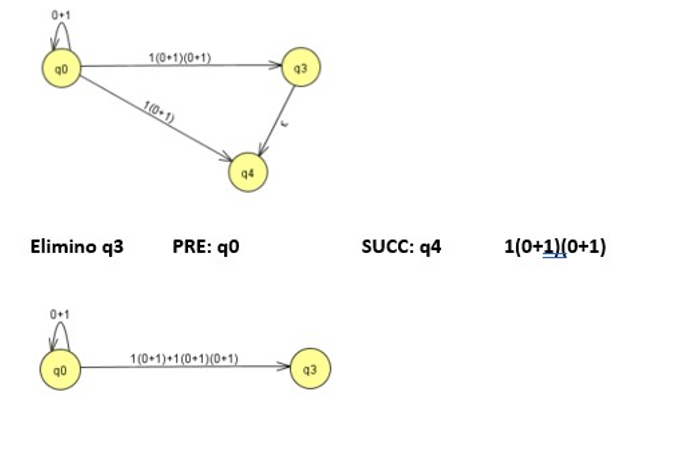


ricordandosi inoltre che:

* se lo stato iniziale presenta più stati entranti verso lo stato iniziale occorre crearne uno nuovo iniziale che non ha archi entranti, aggiungendo una ε-transizione verso il nuovo stato iniziale da parte del vecchio stato iniziale;
* se l’automa presenta più stati finali oppure presenta più transizioni uscenti dallo stato finale, si convertono tutti gli stati finali in stati non-finali, creando un nuovo stato finale e aggiungendo una ε-transizione verso il nuovo stato finale da parte dei veccho stati finali;

Un esempio completo:





Come detto, definiamo linguaggio regolare quello accettabile da u automa.

Esistono però linguaggi in cui alcuni esiste di solito almeno un loop tale per cui, aumentando il numero di stringhe su di esso, il linguaggio si “sbilancia” a favore di una delle parti della stringa, avendo quindi una rappresentazione diversa da quella di partenza.

Viene infatti enunciato il *pumping lemma*, con le seguenti proprietà:

Immagine che contiene testo

Descrizione generata automaticamente

In forma sintetica le condizioni saranno sempre: y ≠ 0 , xy <= p e xyiz ∈ L

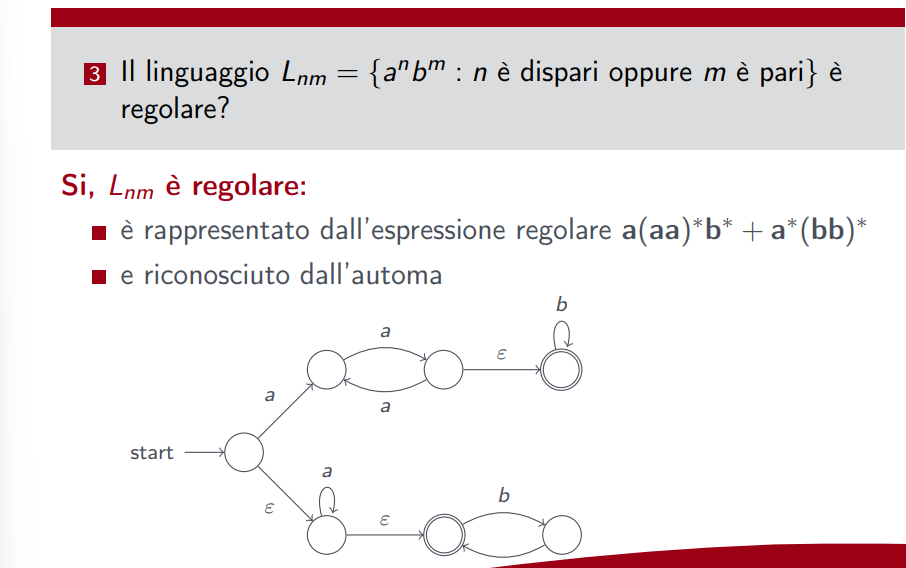
La dimostrazione avviene sempre per contraddizione o per assurdo, tipo così:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente



Possiamo quindi vedere anche il *gioco del pumping lemma*, in cui:

* il giocatore 1 deve dimostrare che il linguaggio è regolare
* il giocatore 2 (noi) deve dimostrare che il linguaggio non è regolare

Immagine che contiene testo

Descrizione generata automaticamente

Esistono poi le *grammatiche context-free/CFG*, generalmente composta nel seguente modo:

Immagine che contiene testo

Descrizione generata automaticamente

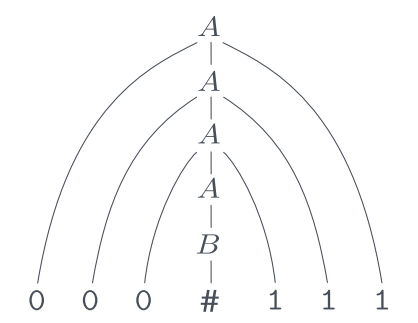
Immagine che contiene testo

Descrizione generata automaticamente

Da una derivazione si costruisce *l’albero sintattico (parse tree)*, che descrive le derivazioni fattibili:

* la radice è la variabile iniziale
* i nodi interni sono variabili
* le foglie sono terminali

Ecco il parse tree per la grammatica sopra:



La grammatica context-free, detta in tre parole, è una grammatica definita ricorsivamente da variabili fino ai terminali, in cui si usano le regole per descrivere un linguaggio con una serie di derivazioni. L’idea quindi è di definire sottostringhe collegate tra di loro ricorsivamente.

Immagine che contiene testo

Descrizione generata automaticamente

Un esempio concreto:

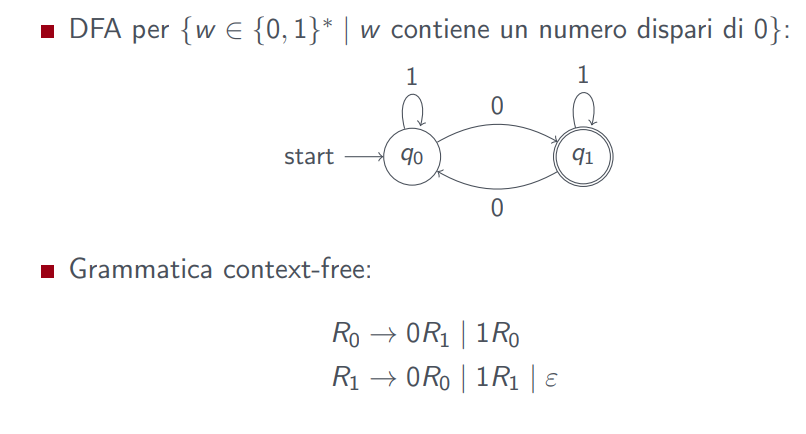
Immagine che contiene testo

Descrizione generata automaticamente

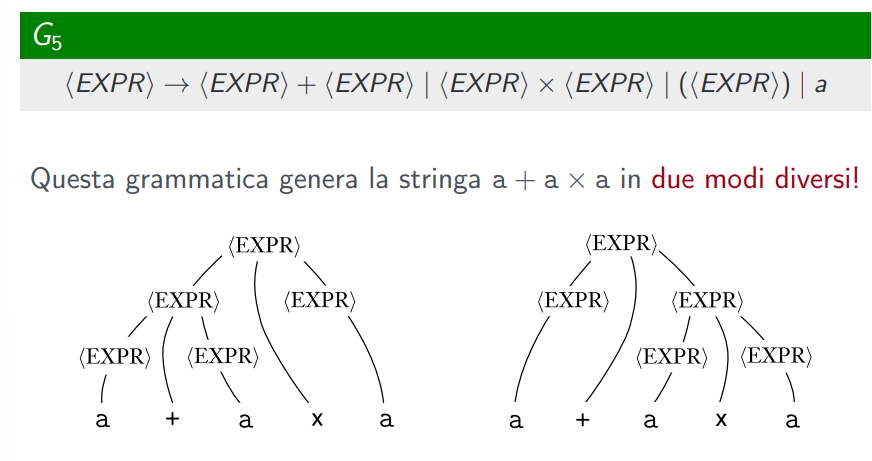
Se il linguaggio è regolare, esiste un DFA che lo riconosce e si segue questa idea con successiva applicazione pratica per *trasformare un DFA in CFG:*

*Immagine che contiene testo

Descrizione generata automaticamente*



Esistono anche le *grammatiche ambigue*, che possono portare a molteplici *leftmost derivations* (quindi più derivazioni a parità di regole partendo da sinistra):



Di solito si usa scrivere la CFG in forma semplificata *(*utile nelle dimostrazioni pratiche in cui si chiede*, dato il linguaggio X che è context-free, dimostra che y è context-free; bisogna infatti trasformare la grammatica di quell’esercizio in Chomsky),* detta *forma normale di Chomsky:*

*Immagine che contiene testo

Descrizione generata automaticamente*

in cui si segue questo ordine di regole:

Immagine che contiene testo

Descrizione generata automaticamente

(dove per la 5 sotto si intende, rimpiazza ogni variabile terminale sul lato destro di una regola con una nuova variabile e regola non terminale)

Quindi in generale:

1)



2)

Immagine che contiene testo

Descrizione generata automaticamente

3)

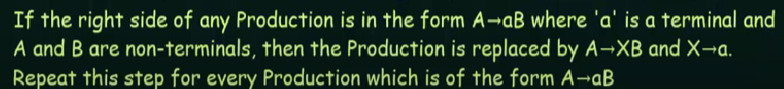
Immagine che contiene testo

Descrizione generata automaticamente

4)



5)



*Esempio completo step by step Chomsky:*

Immagine che contiene testo

Descrizione generata automaticamente

In questo caso si vede che ci sono regole unitarie, ε-simboli e altro.

Per cominciare a trasformarla consideriamo (il testo barrato è quello eliminato):

Immagine che contiene testo

Descrizione generata automaticamente

G’=(V’, Ʃ , R’, S0)

dove si nota che S appare a destra e si introduce un nuovo stato iniziale.

S’ 🡪 S

S 🡪 ASA |aB

A 🡪 B|S|ε

B🡪b|ε

dove metto ε anche in A perché la regola successiva va in B che va a sua volta in ε.

Successivamente:

Immagine che contiene testo

Descrizione generata automaticamente

rimuovo le ε-regole, quindi B 🡪 ε ed A 🡪 ε:

Rimuovendo B 🡪 ε (quindi vuol dire che considero tutte le stringhe dove B è nullo)

S’ 🡪 S

S 🡪 ASA|aB|a

A 🡪 B|S|ε

B🡪b

e poi rimuovo A 🡪 ε (tutti i casi con ASA dove a è nullo e tolgo la ε):

S’ 🡪 S

S 🡪 ASA|aB|a|AS|SA|S

A 🡪B|S

B🡪b

applicando poi la terza parte:

~~Immagine che contiene testo

Descrizione generata automaticamente~~

avendo come regole unitarie (regola che va verso un’altra regola non terminale), quindi:

S🡪S S’🡪S A🡪B A🡪S

Partiamo rimuovendo S 🡪 S, che non fa nulla in pratica:

S’ 🡪 S

S 🡪 ASA|aB|a|AS|SA

A 🡪B|S

B🡪b

quindi eliminiamo S’ 🡪 S (quindi sostituisco S con ASA|aB|a|AS|SA)

S’ 🡪 ASA|aB|a|AS|SA

S 🡪 ASA|aB|a|AS|SA

A 🡪B|S

B🡪b

poi eliminiamo A 🡪 B (quindi sostituisco B con b):

S’ 🡪 ASA|aB|a|AS|SA

S 🡪 ASA|aB|a|AS|SA

A 🡪b|S

B🡪b

ed infine eliminiamo A 🡪 S (quindi sostituisco S con ASA|aB|a|AS|SA):

S’ 🡪 ASA|aB|a|AS|SA

S 🡪 ASA|aB|a|AS|SA

A 🡪b| ASA|aB|a|AS|SA

B🡪b

Immagine che contiene testo

Descrizione generata automaticamente

Ora dobbiamo trovare le produzioni che hanno più di 2 variabili a destra:

S’ 🡪 ASA S 🡪 ASA A 🡪 ASA

In pratica, vedendo che tutte hanno AS oppure SA come variabile rimpiazzabile, posso creare una nuova regola che le sostituisce, ottenendo:

S’ 🡪 AX|aB|a|AS|SA

S 🡪 AX|aB|a|AS|SA

A 🡪 b|AX|aB|a|AS|SA

B🡪 b

X 🡪 SA

Adesso “a” è stato terminale e quindi dobbiamo cambiare tutte le produzioni che contengono “a” con una nuova regola (le espressioni sono S’ 🡪 aB, S 🡪 ab, A 🡪 aB), aggiungendo come regola y 🡪 a:

S’ 🡪 AX|YB|a|AS|SA

S 🡪 AX|YB|a|AS|SA

A 🡪 b|AX|YB|a|AS|SA

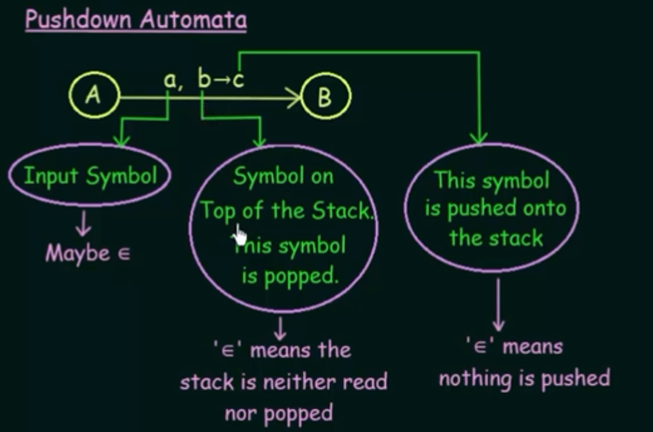
B🡪 b

X 🡪 SA

Y 🡪 a

Passiamo ora agli *automi a pila*, con memoria infinita e che dispongono di operazioni di *push* e *pop*

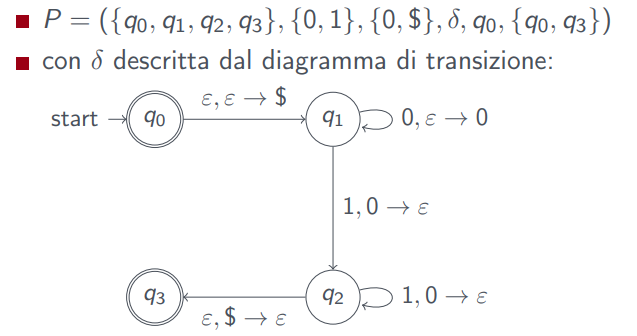
La logica con cui operano è letteralmente descrivibile con questa immagine:



Partendo da una tabella di transizione:

Immagine che contiene tavolo

Descrizione generata automaticamente



Un PDA accetta un linguaggio se:

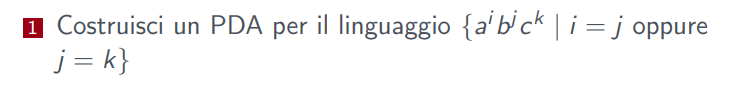
Immagine che contiene testo

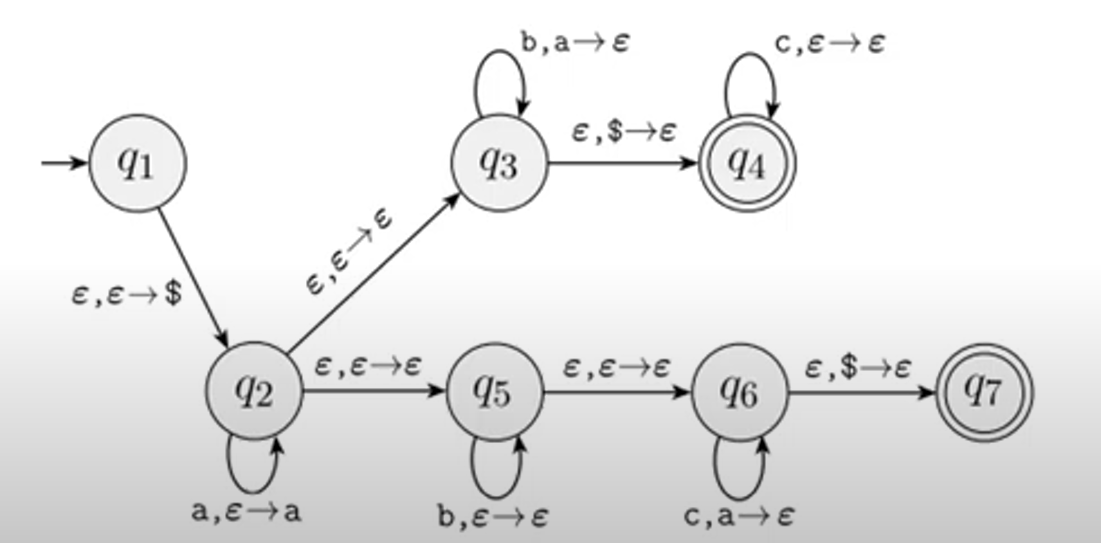
Descrizione generata automaticamente

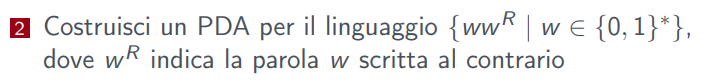
Immagine che contiene testo

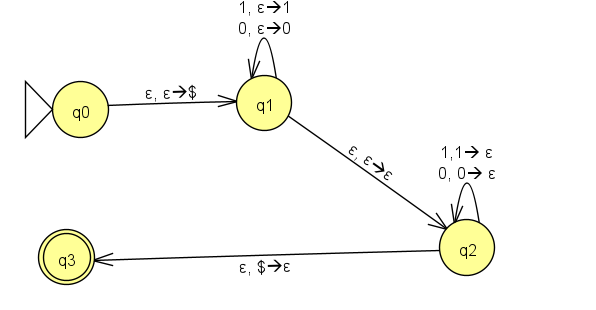
Descrizione generata automaticamente

Vediamo poi due esempi pratici:

**

**

**

**L’osservazione è che, essendo la stringa palindroma, il primo passaggio butta lo stato iniziale che sarà l’ultimo ad essere rimosso, lo stato successivo andrà in se stesso con entrambi 0/1 (permesso perché è un automa non deterministico). A questo punto l’automa controlla idealmente se quello che c’era nello stack è uguale al resto della stringa, poppando a/b se sono nella cima dello stack e poi togliamo lo stato iniziale, accettando la stringa.

Un esempio di stringa accettata: *abba*, in quanto subito *b* è cima dello stack e viene tolto, assieme ad *a*.

Per lo stesso motivo, non viene accettata *abab*, in quanto non si rispetta l’ordine di pop.

Segue il PDA qui a lato: