Linguaggi e Computabilità

UniShare

Davide Cozzi
@dlcgold

Gabriele De Rosa
@derogab

Federica Di Lauro
@f_dila

Indice

1 Introduzione **2**

1.1 Definizioni 2

1.1.1 Alberi Sintatici 14

1.1.2 Grammatiche ambigue 18

1.1.3 Grammatiche Regolari 20

1.1.4 Espressioni Regolari (Regex) 23

1.2 Automi 29

1.2.1 Automi deterministici 30

1.2.2 Automi non deterministici 36

1.2.3 Da espressioni regolari a automi E-NFA 50

1.2.4 Chiusura di un Linguaggio regolare 57

1.2.5 Minimizzazione 62

1.2.6 Pumping Lemma per i linguaggi regolari 69

1.3 Automi a Pila 71

Capitolo 1

Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: `https://github.com/dlcgold/Appunti`.
Grazie mille e buono studio!

1.1 Definizioni

- un **linguaggio** è un insieme di stringhe che può essere generato mediante un dato meccanismo con delle date caratteristiche; un linguaggio può essere riconosciuto, ovvero dando in input una stringa un meccanismo può dirmi se appartiene o meno ad un linguaggio. I meccanismi che generano linguaggi si chiamano *grammatiche*, quelli che li riconoscono *automi*. I linguaggi formali fanno parte dell'informatica teorica *(TCS)*
- si definisce **alfabeto** come un insieme finito e non vuoto di simbolo (come per esempio il nostro alfabeto o le cifre da 0 a 9). Solitamente si indica con Σ o Γ
- si definisce **stringa** come una sequenza finita di simboli (come per esempio una parola o una sequenza numerica). La stringa vuota è una sequenza di 0 simboli, e si indica con ε o λ
- si definisce **lunghezza di una stringa** il numero di simboli che la compone (ovviamente contando ogni molteplicità). Se si ha $w \in \Sigma^*$ è una stringa w con elementi da Σ^* (insieme di tutte le stringhe di tutte le lunghezze possibili fatte da Σ), allora $|w|$ è la lunghezza di w, inoltre $|\varepsilon| = 0$.

- si definisce **potenza di un alfabeto** Σ^k come l'insieme di tutte le sequenze (espressi come stringhe e non simboli) di lunghezza $k \in \mathbb{N}$, $k > 0$ ottenibili da quell'alfabeto (se Σ^2 si avranno tutte le sequenza di 2 elementi etc...). Se ho $k = 1$ si ha $\Sigma^1 \neq \Sigma$ in quanto ora ho stringhe e non simboli. Se ho $k = 0$ ho $\Sigma^0 = \varepsilon$. Dato k ho $|\Sigma|$ che è la cardinalità dell'insieme Σ (e non la sua lunghezza come nel caso delle stringhe); sia $w \in \Sigma^k = a_1, a_2, ..., a_k$, $a_i \in \Sigma$ e $|\Sigma| = q$ ora:
$$|\Sigma^k| = q^k$$
- si definisce Σ^* come **chiusura di Kleene** che è l'unione infinita di Σ^k ovvero
$$\Sigma* = \Sigma^0 \cup \Sigma^1 \cup ... \cup \Sigma^k$$
- si ha che Σ^+ è l'unione per $k \geq 1$ di Σ^k ovvero:
$$\Sigma+ = \Sigma^1 \cup \Sigma^2 \cup ... \cup \Sigma^k = \Sigma^* - \Sigma^0$$
per esempio, per l'insieme $\{0, 1\}$ si ha:
$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 100, 000, ...\}$$
- quindi un **linguaggio** L è un insieme di stringhe e:
$$L \subseteq \Sigma^*$$
si hanno sottoinsiemi particolari, come l'insieme vuoto, che resta però un linguaggio, il **linguaggio vuoto** e $\emptyset \in \Sigma^k$, $|\emptyset| = 0$ che è diverso dal linguaggio che contiene la stringa vuota $|\varepsilon| = 1$ (che conta come una stringa). Inoltre $\Sigma^* \subseteq \Sigma^*$ che ha lunghezza infinita. Posso concatenare due stringhe con un punto: $a \cdot b \cdot c = abc$ e $a \cdot \varepsilon = a$. Ovviamente la stringa concatenata è lunga come la somma delle lunghezze delle stringhe che la compongono. Vediamo qualche esempio di linguaggio:
  - il linguaggio di tutte le stringhe che consistono in n 0 seguiti da n 1:
$$\{\varepsilon, 01, 0011, 000111, ...\}$$
  - l'insieme delle stringhe con un uguale numero di 0 e di 1:
$$\{\varepsilon, 01, 10.0011, 0101.1001, ..\}$$
  - l'insieme dei numeri binari il cui valore è un numero primo:
$$\{\varepsilon, 10, 11, 101, 111, 1011, ...\}$$

- Σ^* è un linguaggio per ogni alfabeto Σ
- $\emptyset$, il linguaggio vuoto, e $\{\varepsilon\}$ sono un linguaggio rispetto a qualunque alfabeto

Prendiamo un alfabeto $\Sigma = \{0, 1\}$ con la sua chiusura di Kleen $\Sigma = \{0, 1\}^*$. Quando si ha un input si può avere un problema di decisione, P, che dia come output "si" o "no". Posso avere un problema di decisione (o *membership*) su $w \in \Sigma = \{0, 1\}^*$, con w stringa, che dia in output "si" o "no". Un linguaggio L sarà:

$$L = \{w \in \{0, 1\}^* \mid P(w) = si$$

quindi si ha che:

$$\Sigma^* \backslash L = \{P(w) = no\}$$

Vediamo ora un esempio di *Context Free Language (CFL)*, costruito a partire da una *Context Free Grammar (CFG)*:

Esempio 1. *Sia* $\Sigma = \{0, 1\}$ *e* L_{pal} = *"stringhe palindrome binarie". Quindi, per esempio,* $0110 \in L$, $11011 \in L$ *ma* $10010 \notin L$. *Si ha che* ε, *la stringa vuota, appartiene a* L. *Diamo una definizione ricorsiva:*

- ***base:*** ε, $0 \;\; 1 \in L_{pal}$
- ***passo:*** *se* w *è palindroma allora* $0w0$ *è palindromo e* $1w1$ *è palindromo*

una variabile generica S *può sottostare alle regole di produzione di una certa grammatica. In questo caso si ha uno dei seguenti:*

$$S \to \varepsilon,\; S \to 0,\; S \to 1,\; S \to 0S0,\; S \to 1S1$$

Si ha che una grammatica G è una quadrupla $G = (V, T, P, S)$ con:

- V simboli variabili
- T simboli terminali, ovvero i simboli con cui si scrivono le stringhe alla fine
- P regole di produzione
- S variabile di partenza *start*

riprendiamo l'esempio sopra:

Esempio 2.

$$G_{pal} = (V = \{S\},\, T = \{0, 1\},\, P,\, S)$$

con:

$$P = \{S \to \varepsilon,\, S \to 0,\, S \to 1,\, S \to 0S0,\, S \to 1S1\}$$

Si può ora costruire un algoritmo per creare una stringa palindroma a partire dalla grammatica G:

$$\underbrace{S}_{\textit{start}} \quad \underbrace{\to}_{\textit{applico una regola}} \quad 1S1 \to 01S10 \to \quad \underbrace{01010}_{\textit{sostituisco variabile}}$$

con S, $1S1$ *e* $01S10$ *che sono forme sentenziali. Posso così ottenere tutte le possibili stringhe. Esiste anche una forma abbreviata:*

$$S \to \varepsilon|o|1|0S0|1S1$$

Non si fanno sostituzioni in parallelo, prima una S *e poi un'altra*

Si hanno 4 grammatiche formali, *gerarchia di Chomsky*:

- **tipo 0:** non si hanno restrizioni sulle regole di produzione, $\alpha \to \beta$. Sono linguaggi ricorsivamente numerabili e sono rappresentati dalle *macchine di Turing*, deterministiche o non deterministiche (la macchina di Turing è un automa)

- **tipo 1:** il lato destro della produzione ha lunghezza almeno uguale a quello sinistro. Sono grammatiche dipendenti dal contesto (*contestuali*) e come automa hanno *la macchina di Turing che lavora in spazio lineare*:
$$\alpha_1 A \alpha_2 \to \alpha_1 B \alpha_2$$
con α_1 e α_2 detti *contesto* e α_1, α_2, $\beta \in (V \cup T)^*$

- **tipo 2:** sono quelle libere dal contesto, context free. Come regola ha $A \to \beta$ con $A \in V$ e $\beta \in V \cup T)^*$ e come automa ha gli *automi a pila non deterministici*

- **tipo 3:** sono le grammatiche *regolari*. Come regole ha $A \to \alpha B$ (o $A \to B\alpha$) e $A \to \alpha$ con $A, B \in V$ e $\alpha \in T$. Come automi ha gli *automi a stato finito deterministici o non deterministici*

Esempio 3. *Sia* $G = (V, T, O, E)$*, con* $V = \{E, I\}$ *e* $T = \{a, b, 0, 1, (,), +, *\}$ *quindi ho le seguenti regole, è di tipo 3:*

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

voglio ottenere $a*(a+b00)$ *sostituisco sempre a destra (right most derivation)*

$$E \to E * E \to E * (E) \to E * (E + E) \to E * (E + I) \to E + (E + I0)$$

$$\to R + (I + b00) \to E * (a + b00) \to I * (a + b00) \to a * (a + b00)$$

usiamo ora l'inferenza ricorsiva:

passo	*stringa ricorsiva*	*var*	*prod*	*passo stringa impiegata*
1	*a*	*I*	*5*	\
2	*b*	*I*	*6*	\
3	*b0*	*I*	*9*	*2*
4	*b00*	*I*	*9*	*3*
5	*a*	*E*	*1*	*1*
6	*b00*	*E*	*1*	*4*
7	*a+b00*	*E*	*2*	*5,6*
8	*(a+b00)*	*E*	*4*	*7*
9	*a*(a+b00)*	*E*	*3*	*5, 8*

definisco formalmente la derivazione $\to$:

Definizione 1. *Prendo una grammatica $G = (V, T, P, S)$, grammatica CFG. Se $\alpha A\beta$ è una stringa tale che $\alpha, \beta \in (V \cup T)^*$, appartiene sia a variabili che terminali. Sia $A \in V$ e sia $a \to \gamma$ una produzione di G. Allora scriviamo:*

$$\alpha A\beta \to \alpha\gamma\beta$$

con $\gamma \in (V \cup T)^*$.
Le sostituzioni si fanno indipendentemente da α e β. Questa è quindi la definizione di derivazione.

Definizione 2. *Definisco il simbolo $\to_*$, ovvero il simbolo di derivazioni in 0 o più passi. Può essere definito in modo ricorsivo. Per induzione sul numero di passi.*

- *la base dice che $\forall\alpha \in (V \cup T)^*$, $\alpha \to *\alpha$*
- *il passo è: se $\alpha \to_G *\beta$ e $\beta \to_G *\gamma$ allora $\alpha \to *\gamma$*

Si può anche dire che $\alpha \to_G *\beta$ sse esiste una sequenza di stringhe $\gamma_1, ..., \gamma_n$ con $n \geq 1$ tale che $\alpha = \gamma_1$, $\beta = \gamma_n$ e $\forall i$, $1 < i < n - 1$ si ha che $\gamma_1 \to \gamma_{i+1}$ la derivazione in 0 o più passi è la chiusura transitiva della derivazione

Definizione 3. *avendo ora definito questi simboli possiamo definire una forma sentenziale. Infatti è una stringa α tale che:*

$$\forall\alpha \in (V \cup T)^* \text{ tale che } S \to_G *\alpha$$

Definizione 4. *data $G = (V, T, P, S)$ si ha che $L(G) = \{w \in T^* |\, S \to_G *w\}$ ovvero composto da stringhe terminali che sono derivabili o 0 o più passi.*

Esempio 4. *formare una grammatica CFG per il linguaggio:*

$$L = \{0^n 1^n | n \geq 1\} = \{01, 0011, 000111, ...\}$$

con x^n intendo una concatenazione di n volte x (che nel nostro caso sono 0 e 1).
posso scrivere:

$$0^n 1^n = 00^{n-1}1^{n-1}1$$

il nostro caso base sarà la stringa 01, Poi si ha: $G = (V, T, P, S)$, $T = \{0, 1\}$, $V = \{S\}$, il caso base $S \to 01$ e $S \to 0S1$ il caso passo è quindi: se $w = 0^{n-1}1^{n-1} \in L$ allora $0w1 \in L$.
Ora voglio dimostare che $000111 \in L$, ovvero $S \to *000111$:

$$S \to 0S1 \to 00S11 \to 000S111$$

Teorema 1. *data la grammatica $G = \{V, T, P, S)$ CFG e $\alpha \in (V \cup T)^*$. Si ha che vale $S \to *\alpha$ sse $S \to_{lm} *\alpha$ sse $S \to_{rm} *\alpha$. Con $\to_{lm} *$ simbolo di left most derivation e $\to_{rm} *$ simbolo di right most derivation*

Esempio 5. *formare una grammatica CFG per il linguaggio:*

$$L = \{0^n 1^n | n \geq 0\} = \{\varepsilon, 01, 0011, 000111, ...\}$$

stavolta abbiamo anche la stringa vuota. Il caso base stavolta è $S \to \varepsilon |\, 0S1$

Esempio 6. *Fornisco una CFG per $L = \{a^n | n \geq 1\} = \{a, aa, aaa, ...\}$. La base è a*
il passo è che se $a^{n-1} \in L$ allora $a^{n-1}a \in L$ (o che $aa^{n-1} \in L$).
Si ha la grammatica $G = \{V, T, P, S)$, $V = \{S\}$, $T = \{a\}$ e si hanno $S \to a |\, Sa$ (o $S \to a |\, aS$). Dimostro che $a^3 \in L$.

$$S \to Sa \to Saa \to aaa$$

oppure

$$S \to aS \to aaS \to aaa$$

Esempio 7. *trovo una CFG per $L = \{(ab)^n | n \geq 1\} = \{ab, abab, ababab, ...\}$*
La base è ab
il passo è che se $(ab)^{n-1} \in L$ allora $(ab)^{n-1}ab \in L$.
Si ha la grammatica $G = \{V, T, P, S)$, $V = \{S\}$, $T = \{a, b\}$ (anche se in realtà $T = \{ab\}$) e si hanno $S \to ab |\, Aab$. Poi dimostro come l'esempio sopra

Esempio 8. *trovo una CFG per $L = \{a^n cb^n | n \geq 1\} = acb, aacbb, aaacbbb, ...\}$*
Il caso base è acb il passo è che se $a^{n-1}cb^{n-1} \in L$ allora $a^{n-1}cb^{n-1}acb \in L$
Si ha la grammatica $G = \{V, T, P, S)$, $V = \{S\}$, $T = \{a, b, c\}$ e si hanno $S \to aSb | acb$.
dimostro che $aaaacbbbbb \in L$:

$$S \to aSb \to aaSbb \to aaaaSbbb \to aaaacbbbb$$

provo a usare anche una grammatica regolare, con le regole $S \to aS | c$, $c \to cB$ e $B \to bB | b$;

$$S \to aS \to aaS \to aaC \to aacB \to aacb...$$

non si può dimostrare in quanto non si può imporre una regola adatta

Esempio 9. *$L = \{a^n cb^{n-1} | n \geq 2\}$, con $a^n cb^{n-1} = a^{n-1}acb^{n-1}$. $S \to aSb | aacb$. Quindi:*

$$S \to aSb \to aaaccbb \in L$$

Esempio 10. *cerco CFG per* $L = \{a^n c^k b^n |\, n,\, k > 0\}$. *a e b devono essere uguali, uso quindi una grammatica context free, mentre c genera un linguaggio regolare.*
Si ha la grammatica $G = \{V, T, P, S)$, $V = \{S, C\}$, $T = \{a, b, c\}$ *e si hanno* $S \to aSb|aCb$ *e* $C \to cC|c$. *dimostro che* $aaaccbbb \in L, n = 3,\, k = 2$:

$$S \to aSb \to aaSbb \to aaaCbbb \to aaacCbbb \to aaaccbbb$$

Esempio 11. *scrivere CFG per* $L = \{a^n b^n c^k b^k |\, n,\, k \geq 0\}$

$$= \{w \in \{a, b, c, d\}^* |\, a^n b^n c^k b^k |\, n,\, k \geq 0\}$$

quindi L concatena due linguaggi L1 e L2, $X = \{a^n b^n\}$ *e* $Y = \{c^k d^k\}$:

$$X \to aXb|\varepsilon$$

$$Y \to cYd|\varepsilon$$

$$S \to XY$$

voglio derivare abcd:

$$S \to XY \to XcYd \to aXbcYd \to aXbc\varepsilon d \to a\varepsilon bc\varepsilon d \to abcd$$

voglio derivare cd

$$S \to XY \to Y \to cYd \to cd$$

Quindi se ho $w \in L1, L2$, ovvero appartenente ad una concatenazione di linguaggi prima uso le regole di un linguaggio, poi dell'altro e infine ottengo il risultato finale.

Esempio 12. *scrivere CFG per* $L = \{a^n b^k c^k d^n |\, n > 0,\, k \geq 0\}$.

$$S \to aSd|\, aXd$$

$$X \to bXc|\varepsilon$$

derivo aabcdd:

$$S \to aSd \to aaXdd \to aabXcdd \to aabcdd$$

Esempio 13. *scrivere CFG per* $L = \{a^n c b^n c^m a d^m |\, n > 0,\, m \geq 1\}$.

$$S \to XY$$

$$X \to aXb|c$$

$$Y \to cUd|cad$$

$$S \to XY \to cY \to ccad$$

Esempio 14. *scrivere CFG per* $L = \{a^{n+m}xc^nyd^m | \, n, \, m \geq 0\}$*.* $a^{n+m} = a^na^m$ *o* a^ma^n*. Si hanno 2 casi:*

1. $L = \{a^na^mxc^nyd^m | \, n, \, m \geq 0\}$
2. $L = \{a^ma^nxc^nyd^m | \, n, \, m \geq 0\}$

ma solo $L = \{a^ma^nxc^nyd^m | \, n, \, m \geq 0\}$ *può generare una CFG (dove non si possono fare incroci, solo concatenazioni e inclusioni/innesti).*

$$S \to aSd|Y$$

$$Y \to Xy$$

$$X \to aXc|x$$

si può fare in 2:

$$S \to aSd|Xy$$

$$X \to aXc|x$$

derivo con $m = n = 1$*,* $aaxcyd$*:*

$$S \to aSd \to aXyd \to aaXcyd \to aaxcyd$$

Esempio 15. *scrivere CFG per* $L = \{a^nb^m | \, n \geq m \geq 0\}$*.*

$$L = \{\varepsilon, a, ab, aa, aab, aabb, aaa, aaab, aaabb, aaabbb, ...\}$$

Se $n \geq m$ *allora* $\exists k \geq 0 \to n = m + k$*. Quindi:*

$$l = \{a^{m+k}b^m|m, k \geq 0\}$$

si può scrivere in 2 modi:

1. $l = \{a^ma^kb^m|m, k \geq 0\}$ *quindi con innesto*
2. $l = \{a^ka^mb^m|m, k \geq 0\}$*quindi con concatenazione*

entrambi possibili per una CFG:

1.

$$S \to XY$$

$$X \to aX|\varepsilon \text{ si può anche scrivere } X \to Xa|\varepsilon$$

$$Y \to aYb|\varepsilon$$

oppure

$$S \to aS|X$$

$$X \to aXb|\varepsilon$$

2.
$$S \to aSb|\varepsilon$$
$$X \to aX|\varepsilon$$

Esempio 16. *scrivere CFG per* $L = \{a^n b^{m+n} c^h |\, m > h \geq 0,\, n \geq 0\}$.
Se $n > h$ *allora* $\exists k \to n = h + k$*, quindi:*

$$L = \{a^n b^{m+h+k} c^h |\, m > h \geq 0,\, n \geq 0\}$$

. *ovvero:*
$$L = \{a^n b^n b^k b^h c^h |\, m \geq 0, k > 0, h \geq 0\}$$

si ha:
$$S \to XYZ$$
$$X \to aXb|\varepsilon$$
$$Y \to Yb|b$$
$$Z \to bZc|\varepsilon$$

si può anche fare:
$$S \to XY$$
$$X \to aXb|\varepsilon$$
$$Y \to bYc|Z$$
$$Z \to bZ|b$$

Esempio 17. *scrivere CFG per* $L = \{a^n b^m c^k |\, k > n + m,\, n, m \geq 0\}$.
per $n = m = 0$, $k = 1$ *avrò la stringa* c. *se* $k > n + m$ *allora* $\exists l > 0 \to k = n + m + l$ *quindi:*

$$L = \{a^n b^m c^{n+m+l} |\, l > 0,\, n, m \geq 0\}$$
$$= L = \{a^n b^m c^n c^m c^l |\, l > 0,\, n, m \geq 0\}$$

sistemando:
$$= L = \{a^n b^m c^l c^m cnl |\, l > 0,\, n, m \geq 0\}$$

quindi:
$$S \to aSc|X$$
$$X \to bXc|Y$$
$$Y \to cY|c$$

Esempio 18. *scrivere CFG per* $L = \{a^n x c^{n+m} y^h z^k d^{m+h} | \, n, m, k, h \geq 0\}$. *ovvero:*

$$L = \{a^n x c^n c^m y^h z^k d^h d^m | \, n, m, k, h \geq 0\}$$

quindi avrò:

$$S \to XY$$

$$X \to aXc|x$$

$$Y \to cYd|W$$

$$W \to yWd|X$$

$$Z \to zZ|\varepsilon$$

Esempio 19. *vediamo un esempio di grammatica dipendente dal contesto:*

$$L = \{a^n b^n c^n | \, n \geq 1\}$$

$G = \{V, T, P, S\} = \{(S, B, C, X)\} = \{(a, b, c), P, S\}$ *ecco le regole di produzione (qui posso scambiare variabili a differenza delle context free):*

1. $S \to aSBC$
2. $S \to aBC$
3. $CB \to XB$
4. $XB \to XC$
5. $XC \to BC$
6. $aB \to ab$
7. $bB \to bb$
8. $bC \to bc$
9. $cC \to cc$

vediamo un esempio di derivazione: per $n = 1$ *ho abc ovvero:*

$$S \to aBC \to abC \to abc$$

con $n = 2$ *ho aabbcc:* $S \to aSBC \to aaBCBC \to aaBXBC \to aaBXCC \to aaBBCC \to aabBCC \to aabbCC \to aabbcC \to aabbcc$

Esempio 20. *vediamo un esempio di grammatica dipendente dal contesto:*

$$L = \{a^n b^m c^n d^m \,|\, n, m \geq 1\}$$

Si ha:

$$G = (\{S, X, C, D, Z\}, \{a, b, c, d\}, P, S)$$

con le seguenti regole di produzione:

- $S \to aSc|\, aXc$
- $X \to bXD|\, bD$
- $DC \to CD$
- $DC \to DZ$
- $DZ \to CZ$
- $XZ \to CD$
- $bC \to bc$
- $cC \to cc$
- $cD \to cd$
- $dD \to dd$

provo a derivare aabbbccddd quindi con $n = 2$*,* $m = 3$*:*

$$S \to aSC \to aaXCC \to aabXDCC \to aabbXDDCC \to$$
$$aabbbDDDCC \to aabbbCCDDD \to aabbbccddd$$

Esempio 21. *Sia* $L = \{w \in \{a, b\}^*|$ *w contiene lo stesso numero di a e b*$\}$*:*

$$S \to aSbS|\, bSaS|\, \varepsilon$$

dimostro per induzione che è corretto:

- ***caso base:*** $|w| = 0 \to w = \varepsilon$

- ***caso passo:*** *si supponga che G produca tutte le stringhe (di lunghezza $<$ di n) di $\{a, b\}^*$ con lo stesso numero di a e b e dimostro che produce anche quelle di lunghezza n, sia:*

  $w \in \{a, b\}^* \mid |w| = n$ *con a e b in egual numero,* $m(a) = m(b)$ *con m() che indica il numer*

  quindi si ha che:

  $$w = aw_1bw_2 \text{ o } w = bw_1aw_2$$

  sia.

  $$k_1 = m(a) \in w_1 = m(b) \in w_1$$

  $$k_2 = m(a) \in w_2 = m(b) \in w_2$$

  allora:

  $$k_1 + k_2 + 1 = m(a) \in w = m(b) \in W$$

  sapendo che $|w_1| < n$ e $|w_2| < n$ allora w_1 e w_2 sono egnerati da G per ipotesi induttiva

1.1.1 Alberi Sintatici

Definizione 5. *Data una grammatica CFG, $G = \{V, T, P, S\}$ un **albero sintattico** per G soddisfa le seguenti condizioni:*

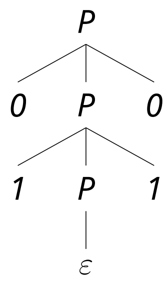
- *ogni nodo interno è etichettato con una variabile*
- *ogni foglia è anch'essa etichettata con una variabile o col simbolo di terminale T o con la stringa vuota ε (in questo caso la foglia è l'unico figlio del padre)*
- *se un nodo interno è etichettato con A i suoi figli saranno etichettati con X1, ..., Xk e $A \to X1, ..., Xk$ sarà una produzione di G. Se un Xi è ε sarà l'unica figlio e $A \to \varepsilon$ sarà comunque una produzione di G*

*La concatenazione in ordine delle foglie viene detto **prodotto dell'albero***

Esempio 22. *Usiamo l'esempio delle stringhe palindrome:*

$$P \to 0P0|\, 1P1|\varepsilon$$

sia il seguente albero sintatico:

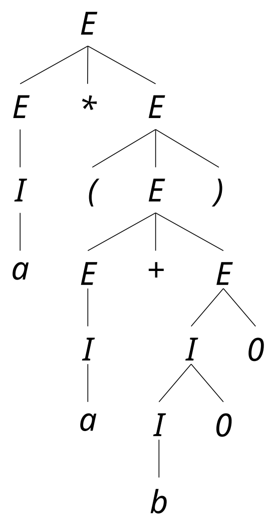


Esempio 23. *Si ha:*

$$E \to I|\, E + E|\, E * E|\, (E)$$

$$I \to a|\, b|\, Ia|\, Ib|\, I0|\, I1$$

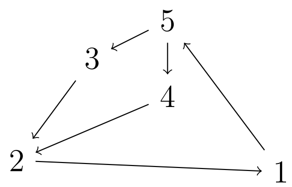
un albero sintattico per $a * (a + b00)$ *può essere:*



Data una CFG si ha che i seguenti cinque enunciati si equivalgono:

1. la procedura di inferenza ricorsiva stailisce che una stringa w di simboli terminali appartiene al linguaggio $L(A)$ con A variabile
2. $A \to^* w$
3. $A \to^*_{lm} w$
4. $A \to^*_{rm} w$
5. esiste un albero sintattico con radice A e prodotto w

queste 5 proposizioni si implicano l'uni l'altra:



vediamo qualche dimostrazione di implicazione tra queste proposizioni:

da 1 a 5. si procede per induzione:

- **caso base:** ho un livello solo (una sola riga), $\exists A \to w$:

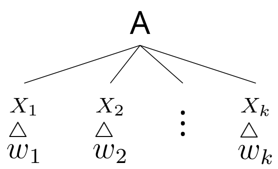
$$\begin{matrix} A \\ \triangle \\ w \end{matrix}$$

- **caso passo:** suppongo vero per un numero di righe $\leq n$, lo dimsotro per $n+1$ righe:

$$A \to X_1, X_2, ..., X_k$$

$$w = w_1, w_2, ..., w_k$$

ovvero, in meno di $n+1$ livelli:



□

da 5 a 3. procedo per induzione:

- **caso base (n=1):** $\exists A \to w$ quindi $A \to_{lm} w$, come prima si ha un solo livello:

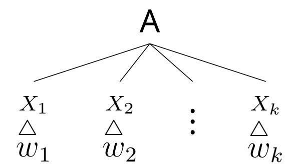
$$\overset{A}{\underset{w}{\triangle}}$$

- **caso passo:** suppongo che la proprierà valga per ogni albero di profondità minore uguale a n, dimostro che valga per gli alberi profondi $n+1$:

$$A \to X_1, X_2, ..., X_k$$

$$w = w_1, w_2, ..., w_k$$

ovvero, in meno di $n+1$ livelli:



$$A \to_{lm} X_1, X_2, ..., X_k$$

$$x_1 \to^*_{lm} w_1 \text{ per ipotesi induttiva si ha un albero al più di n livelli}$$

quindi:

$$A \to_{lm} X_1, ..., X_k \to^*_{lm} w_1, X_2, ..., X_k \to^*_{lm} ... \to^*_{lm} w_1, ..., w_k = w$$

Esempio 24.

$$E \to I \to Ib \to ab$$

$$\alpha E\beta \to \alpha I\beta \to \alpha Ib\beta \to \alpha ab\beta, \ \ \alpha, \beta \in (V \cup T)^*$$

□

Esempio 25. *Mostro l'esistenza di una derivazione sinistra dell'albero sintattico di* $a * (a + b00)$*:*

$$E \to^*_{lm} E * E \to^*_{lm} I * E \to^*_{lm} a * E \to^*_{lm} a * (E) \to^*_{lm} a * (E + E) \to^*_{lm}$$

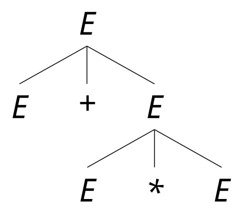
$$a*(I+E) \to^*_{lm} a*(a+E) \to^*_{lm} a*(a+I) \to^*_{lm} a+(a+I0) \to^*_{lm} a*(a+I00) \to^*_{lm} a*(a+b00)$$

1.1.2 Grammatiche ambigue

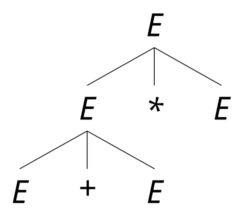
Definizione 6. *Una grammatica è definita ambigua se esiste una stringa w di terminali che ha più di un albero sintattico*

Esempio 26. *vediamo un esempio:*

1. $E \to E + E \to E + E * E$ *ovvero:*



2. $E \to E * E \to E + E * E$ *ovvero:*



si arriva a due stringhe uguali ma con alberi diversi. Introduciamo delle categorie sintatiche, dei vincoli alla produzione delle regole:

1. $E \to T|\, E + T$
2. $T \to F|\, T + F$
3. $F \to I|\, (E)$
4. $I \to a|\, b|\, Ia|, Ib|\, I0|\, I1$

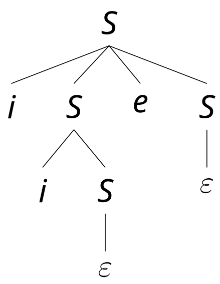
Possono esserci più derivazioni di una stringa ma l'importante è che non ci siano alberi sintattici diversi. Capire se una CFG è ambigua è un problema indecidibile

Esempio 27. *vediamo un esempio:*

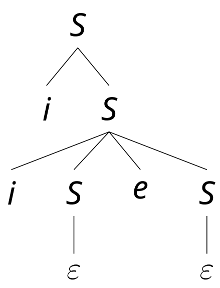
$$S \to \varepsilon|\, SS|\, iS|\, iSeS$$

con S=statement, i=if e e=else. Considero due derivazioni:

1. *$S \to iSeS \to iiSeS \to iie$:*



2. *$S \to iS \to iiSeS \to iieS \to iie$:*



Si ha quindi una grammatica ambigua

Teorema 2. *Per ogni CFG, con $G = (V, T, P, S)$, per ogni stringa w di terminali si ha che w ha due alberi sintattici distinti sse ha due derivazioni sinistre da S distinte.*
Se la grammatica non è ambigua allora esiste un'unica derivazione sinistra da S

Linguaggi inerentemente ambigui

Definizione 7. *Un linguaggio L è inerentemente ambiguo se tutte le grammatiche CFG per tale linguaggio sono a loro volta ambigue*

Esempio 28. *Sia $L = \{a^nb^nc^md^m |\, n, m \geq 1\} \cup \{a^nbmnc^md^n |\, n, m \geq 1\}$*
si ha quindi un CFL formato dall'unione di due CFL. L è inerentemente ambiguo e generato dalla seguente grammatica:

- $S \to AB|C$

- $A \to aAb|\, ab$
- $B \to cBd|\, cd$
- $C \to aCd|\, aDd$
- $D \to bDc|\, bc$

si possono avere due derivazioni:

1. $S \to_{lm} AB \to_{lm} aAbB \to_{lm} aabbB \to_{lm} aabbcBd \to_{lm} aabbccdd$
2. $S \to_{lm} C \to_{lm} aCd \to_{lm} aaBdd \to_{lm} aabBcdd \to_{lm} aabbccdd$

a generare problemi sono le stringhe con n=m perché possono essere prodotte in due modi diversi da entrambi i sottolinguaggi. Dato che l'intersezione tra i due sottolinguaggi non è buota si ha che L è ambiguo

1.1.3 Grammatiche Regolari

Sono le grammatiche che generano i linguaggi regolari (quelli del terzo tipo) che sono casi particolari dei CFL.
Si ha la solita grammatica $G = (V, T, P, S)$ con però vincoli su P:

- ε si può ottenere solo con $S \to \varepsilon$
- le produzioni sono tutte lineari a destra ($A \to aA$ o $A \to a$) o a sinistra ($A \to Ba$ o $A \to a$)

Esempio 29. *$I \to a|\, b|\, Ia|\, Ib|\, I0|\, I1$ è una grammatica con le produzioni lineari a sinistra.*
Potremmo pensarlo a destra $I \to a|\, b|\, aI|\, bI|\, 0I|\, 1I$.
Vediamo esempi di produzione con queste grammatiche:

- *con $I \to a|\, b|\, Ia|\, Ib|\, I0|\, I1$ possiamo derivare $ab01b0$:*

$$I \to I0 \to Ib0 \to I1b0 \to I01b0 \to Ib01b0 \to ab01b0$$

- *con $I \to a|\, b|\, aI|\, bI|\, 0I|\, 1I$ invece non riusciamo a generare nulla:*

$$I \to 0I \to 0a$$

definisco quindi un'altra grammatica (con una nuova categoria sintattica):

$$I \rightarrow aJ|\, bJ$$

$$J \rightarrow a|\, b|\, aJ|\, bJ|\, 0J|\, 1J$$

che però non mi permette di terminare le stringhe con 0 e 1, la modifico ancora otterdendo:

$$I \rightarrow aJ|\, bJ$$

$$J \rightarrow a|\, b|\, aJ|\, bJ|\, 0J|\, 1J|\, 0|\, 1$$

e questo è il modo corretto per passare da lineare sinistra a lineare destra

Esempio 30. *Sia $G = (\{S\}, \{0,1\}, P, S)$ con $S \rightarrow \varepsilon|\, 0|\, 1|\, 0S|\, 1S$. Si ha quindi:*

$$L(G) = \{0,1\}^*$$

si hanno comunque due proposizioni ridondanti, riducendo trovo:

$$S \rightarrow \varepsilon|\, 0S|\, 1S$$

con solo produzioni lineari a destra. Con produzioni lineari a sinistra ottengo:

$$S \rightarrow \varepsilon|\, S0|\, S1$$

Esempio 31. *Trovo una grammatica lineare destra e una sinistra per $L = \{a^n b^m |\, n, m \geq 0\}$:*

- ***lineare a destra:*** *si ha $G = (\{S, B\}, \{a, b\}, P, S)$ e quindi:*

  $$S \rightarrow \varepsilon|\, aS|\, bB$$

  $$B \rightarrow bB|\, b$$

  ma non si possono generare stringhe di sole b, infatti:

  $$S \rightarrow aS \rightarrow abB \rightarrow abbB \rightarrow abbb$$

  *ma aggiungere ε a B **non è lecito**. posso però produrre la stessa stringa da due derivazioni diverse:*

  $$S \rightarrow \varepsilon|\, aS|\, bB|\, b$$

  $$B \rightarrow bB|\, b$$

  che risulta quindi la nostra lineare a destra

- ***lineare a sinistra:*** *si ha* $G = (\{S, A\}, \{a, b\}, P, S)$ *e quindi:*

$$S \to \varepsilon |\, Sb |\, Ab |\, a$$

$$A \to Aa |\, a$$

Esempio 32. *Trovo una grammatica lineare destra e una sinistra per* $L = \{ab^n cd^m e |\, n \geq 0\,, m > 0\}$*:*

- ***lineare a destra:*** *si ha si ha* $G = (\{S, A, B, E\}, \{a, b, c, d, e\}, P, S)$ *e quindi:*

$$S \to aA$$

$$A \to bA |\, cB$$

$$B \to dB |\, dE$$

$$E \to e$$

- ***lineare a sinistra:*** *si ha si ha* $G = (\{S, X, Y, Z\}, \{a, b, c, d, e\}, P, S)$ *e quindi:*

$$S \to Xe$$

$$A \to Xd |\, Yd$$

$$B \to Zc$$

$$E \to a |\, Zb$$

quindi se per esempio ho la stringa "ciao" si ha:

- ***lineare a destra:***

$$S \to Ao$$

$$A \to Ba$$

$$B \to Ei$$

$$E \to c$$

- ***lineare a sinistra:***

$$S \to cA$$

$$A \to iB$$

$$B \to aE$$

$$E \to o$$

Esempio 33. *A partire da* $G = (\{S, T\}, \{0, 1\}, P, S)$ *con:*

$$S \to \varepsilon |\, 0S |\, 1T$$

$$T \to 0T |\, 1S$$

trovo come è fatto $L(G)$*:*

$$L(G) = \{w \in \{0,1\}^* |\, w \textit{ ha un numero di 1 pari}\}$$

Esempio 34. *fornire una grammatica regolare a destra e sinistra per:*

$$L = \{w \in \{0,1\}^* |\, w \textit{ ha almeno uno 0 o almeno un 1}\}$$

Si ah che tutte le stringhe tranne quella vuota ciontengono uno 0 o un 1 quindi $G = (\{S\}, \{0, 1\}, P, S)$*:*

- ***lineare a destra:***
$$S \to 0 |\, 1 |\, 0S |\, 1S$$
- ***lineare a sinistra:***
$$S \to 0 |\, 1 |\, S0 |\, S1$$

1.1.4 Espressioni Regolari (Regex)

le regex sono usate per la ricerca di un pattern in un testo o negli analizzatori lessicali. Una regex denota il linguaggio e non la grammatica. Si hanno le seguenti operazioni tra due linguaggi L e M:

- **unione:** dati L, $M \in \Sigma^*$, l'unione $L \cup M$ è l'insieme delle stringhe che si trovano in entrambi i linguaggi o solo in uno dei due

  Esempio 35.
  $$L = \{001, 10, 111\}$$
  $$M = \{\varepsilon, 001\}$$
  $$L \cup M = \{\varepsilon, 01, 10, 111, \varepsilon\}$$

  si ha che:
  $$L \cup M = M \cup L$$

- **concatenazione:** dati L, $M \in \Sigma^*$, la concatenazione $L \cdot M$ (o LM) è lisieme di tutte le stringhe ottenibili concatenandone una di L a una di M

Esempio 36.

$$L = \{001, 10, 111\}$$
$$M = \{\varepsilon, 001\}$$
$$L \cdot M = \{001, 001001, 10, ...\}$$

si ha che:

$$L \cdot M \neq M \cdot L$$

- si definiscono:
  - $L \cdot L = L^2$, $L\ cdotL \cdot L = L^3$ etc...
  - $L^1 = L$
  - $L^0 = \{\varepsilon\}$
- **chiusura di Kleene:** dato $L \subseteq \Sigma^*$ si ha che la chiusura di Kleen di L è:

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

ricordando che $l^0 = \varepsilon$

Esempio 37. *Sia* $L = \{0, 11\}$*, si ha:*

$$L^0 = \varepsilon$$

$$L^1 = L = \{0, 11\}$$
$$L^2 = L \cdot L = \{00, 011, 110, 1111\}$$
$$L^3 = L \cdot L \cdot L = L^2 \cdot L = \{000, 0110, 1100, 11110, 0011, 01111, 11011, 111111\}$$

vediamo dei casi particolari:

- $L = \{0^n | \, n \geq 0\}$ implica $|L| = \infty$ e quindi, essendo $L^i = L$, $i \geq 1$ e quindi $|L^i| = \infty$, $|L^*| = \infty$. Si ha quindi:

$$L^* = L^0 \cup L^1 \cup ... \cup L^i = L$$

- $L = \emptyset$ implica $L^0 = \{\varepsilon\}$, $L^2 = L \cdot L = \emptyset$ e così via per ogni concatenazione di L. Si ha quindi:

$$L^* = L^0 = \{\varepsilon\}$$

– $L = \{\varepsilon\}$ implica $L^0 = \{\varepsilon\} = L = L^1 = L^2 = ...$, si ha quindi:

$$L^* = \{\varepsilon\} = L$$

L'insieme vuoto e l'insieme contenente la stringa vuota hanno le uniche chiusure di kleene finite

Definizione 8. *Si riporta la definizione ricorsiva di un'espressione regolare:*

- ***casi base:*** *si hanno tre casi base:*
  1. ε *e* $\emptyset$ *sono espressioni regolari*
  2. *se* $a \in \Sigma$ a *è un'esprssione regolare,* $L(a) = \{a\}$
  3. *le variabili che rappresentano linguaggi regolari sono espressioni regolari,* $L(L) = L$
- ***casi passo:*** *si hanno i 4 casi passo:*
  1. ***unione:*** *se* E *e* F *sono espressioni regolari allora anche* $E+F = E \cup F$ *è un'espressione regolare e si ha:*
     $$L(E+F) = L(E) \cup L(F)$$
  2. ***concatenazione:*** *se* E *e* F *sono espressioni regolari allora anche* $EF = E \cdot F$ *è un'espressione regolare e si ha:*
     $$L(EF) = L(E) \cdot L(F)$$
  3. ***chiusura:*** *se* E *è un'espressione regolare allora* E^* *è un'espressione regolare e si h:*
     $$L(E^*) = (L(E))^*$$
  4. ***parentesi:*** *se* E *è un'espressione regolare allora* (E) *è un'espressione regolare e si ha:*
     $$L((E)) = L(E)$$

Esempio 38. *trovo regex per l'insieme di stringhe in* $\{0,1\}^*$ *che consistono in 0 e 1 alternati:*

$$01 \to \{01\}$$
$$(01)^* \to \{\varepsilon, 01, 0101, 010101, ...\}$$
$$(01)^* + (10)^* \to \{\varepsilon, 01, 10, 0101, 1010, ...\}$$

ma posso volere diverse quantità di 0 e 1, sempre mantenendo l'alternanza, metto o uno 0 o un 1 davanti a quanto ottenuto appena sopra:

$$(01)^* + (10)^* + 0(10)^* + 1(01)^* \to \{\varepsilon, 01, 10, 010, 101, ...\}$$

non è comunque l'unica soluzione, si può avere:

$$(\varepsilon + 1)(01)^*(\varepsilon + 0) \to \{\varepsilon, 01, 10, 010, 101, ...\}$$

oppure ancora:

$$(\varepsilon + 0)(10)^*(\varepsilon + 1)$$

Si ha una precedenza degli operatori, in ordine di precedenza (si valuta da sinistra a destra):

1. chiusura di Kleene *
2. concatenazione $\cdot$, che è associativo ($(E \cdot F) \cdot G = E \cdot (F \cdot G)$) ma non è commutativo ($E \cdot F \neq F \cdot E$)
3. unione $+$ che è associativa ($(E+F)+G = E+(F+G)$) ed è commutativo ($E + F = F + E$)
4. infine le parentesi

si hanno anche delle proprietà algebriche:

- due espressioni regolari sono equivalenti se denotano le stesso linguaggio
- due espressioni regolari con variaboli sono equivalenti se lo sono $\forall$ assegnamento alle variabili
- l'unione è commutativa e associativa, la concatenazione è solo associativa
- si definiscono:

  - **identità:** ovvero un valore unito all'identità è pari a se stesso (elemento neutro della somma $0+x=x+0=x$). $\emptyset$ è identità per l'unione ($\emptyset+L=L+\emptyset=L$), $\{\varepsilon\}$ è identità per la concatenazione ($\varepsilon L=L\varepsilon=L$)
  - **annichilitore:** ovvero un valore concatenato all'annichilatore da l'annichilitore (l'elemento nullo del prodotto $0x=x0=0$). $\emptyset$ è l'annichilitore per la concatenazione ($\emptyset L=L\emptyset=\emptyset$)

- **distributività:** dell'unione rispetto alla concatenazione (che non è commutativa):
  - **distributività sinistra:** $L(M+N)=LM+LN$
  - **distributività destra:** $(M+N)L=ML+NL$
- **idempotenza:** $L+L=L$
- $(L^*)^*=L^*$
- $\emptyset^*=\varepsilon$ infatti $L(\emptyset)=\{\varepsilon\}\cup L(\emptyset)\cup L(\emptyset)\cdot L(\emptyset)\cup ...=\{\varepsilon\}\cup\emptyset\cup\emptyset...=\varepsilon$
- $\varepsilon^*=\varepsilon$ infatti $L(\varepsilon^*)=\{\varepsilon\}\cup L(\varepsilon)\cup L(\varepsilon)=\{\varepsilon\}\cup\{\varepsilon\}\cup...=\{\varepsilon\}=L(\varepsilon)$
- $L^+=L\cdot L^*=L^*\cdot L$ (quindi con almeno un elemento che non sia la stringa vuota)
- $L^*=l^++\varepsilon$

Esempio 39. *Ho* $ER=(0+1)^*0^*(01)^*$*:*

- *001 fa parte del linguaggio? Si:* $\varepsilon\cdot 0\cdot 01$
- *1001 fa parte del linguaggio? Si:* $1\cdot 0\cdot 01$
- *0101 fa parte del linguaggio? Si:* $\varepsilon\cdot\varepsilon\cdot 0101$
- *0 fa parte del linguaggio? Si:* $\varepsilon\cdot 0\cdot\varepsilon$
- *10 fa parte del linguaggio? Si:* $1\cdot 0\cdot\varepsilon$

$L((0+1)^*)=(L(0+1))^*=(L(0)+L(1))^*=(\{0\}\cup\{1\})^*=(\{0,1\})^*=\{0,1\}^*$ *ovvero tutte le combinazioni di 0 e 1*

Si ricorda che:

$$(0+1)^*\neq 0^*+1^*$$

Esempio 40. *ho* $ER = ((01)^* \cdot 10 \cdot (0+1)^*)^*$

- *0101 fa parte del linguaggio? No*
- *01000 fa parte del linguaggio? No*
- *01011 fa parte del linguaggio? No*
- *10111 fa parte del linguaggio? Si,* $\varepsilon \cdot 10 \cdot 111$
- *101010 fa parte del linguaggio? Si, prendo* $10 \cdot 1010$
- *101101 fa parte del linguaggio? Si,* $\varepsilon \cdot 10 \cdot 1$ *due volte*
- *0101100011 fa parte del linguaggio? Si,* $0101 \cdot 10 \cdot 0011$ *(0011 lo posso prendere da* $(0+1)^*$*)*

Esempio 41. *ho* $ER = ((01)^* \cdot 10 \cdot (0+1))^*$

- *0101 fa parte del linguaggio? No*
- *01000 fa parte del linguaggio? No*
- *01011 fa parte del linguaggio? No*
- *10111 fa parte del linguaggio? No*
- *101010 fa parte del linguaggio? No*
- *101101 fa parte del linguaggio? Si,* $\varepsilon \cdot 10 \cdot 1$ *due volte*
- *0101100011 fa parte del linguaggio? No*

Esempio 42. *Da* $L \subseteq \{0,1\}|$ *stringhe contenenti almeno una volta 01 quindi:*

$$(0+1)^*01(0+1)^*$$

Esempio 43. *ho* $ER = (00^*1^*)^*$*, quindi:*

$$L = \{\varepsilon, 0, 01, 000, 001, 010, 011\} = \{\varepsilon\} \cup \{w \in \{0,1\}^* | \text{ } w \textit{ che inizia con 0}\}$$

Esempio 44. *ho* $ER = a(a+b)^*b$*, quindi:*

$$L = \{w \in \{a,b\}^* | \text{ } w \textit{ inizia con } a \textit{ e termina con } b\}$$

Esempio 45. *ho* $ER = (0^*1^*)^*000(0+1)^*$*, quindi, sapendo che* $\{0,1\}^*$ *mi permette tutte le combinazioni che voglio come* $(0+1)^*$*:*

$$L = \{w \in \{0,1\}^* | \text{ } w \textit{ come voglio con tre 0 consecutivi}\}$$

Esempio 46. *ho* $ER = a(a+b)^*c(a+b)^*c(a+b)^*b$, *quindi:*

$$L = \{w \in \{a,b,c\}^* | \text{ w inizia con a, termina con b e contiene almeno due c, eventualmente non adiacenti}\}$$

Esempio 47. *Da* $L \subseteq \{0,1\}|$ *ogni 1 è seguito da 0, a meno che non sia l'ultimo carattere, ovvero 11 non compare quindi:*

$$(10+0)^*(\varepsilon+1)^*$$

Esempio 48. *cerco ER per* $L \subseteq \{0,1\}^*|$ *stringhe contenenti un numero pari di 1:*

$$(0^*10^*1)^*0^*$$

oppure:

$$(0+10^*1)^*$$

Esempio 49. *cerco ER per* $L \subseteq \{0,1\}^*|$ *stringhe contenenti un numero dispari di 1:*

$$(0^*10^*)^*0^*10^*$$

oppure:

$$(0+10^*1)^*10^*$$

Esempio 50. *cerco ER per* $L \subseteq \{0,1\}^*|$ *stringhe contenenti un numero divisibile per 3 di 0:*

$$(1^*01^*01^*0)^*1^*$$

Esempio 51. *cerco ER per* $L \subseteq \{0,1\}^*|$ *stringhe contenenti al più una coppia di 1 consecutivi:*

$$(10+0)^*(11+1+\varepsilon)(01+0)^*$$

Esempio 52. *cerco ER per* $L \subseteq \{a,b,c\}^*|$ *stringhe contenenti almeno una* a *e almeno una* b :

$$c^*\left(a(a+c)^*b+b(b+c)^*a\right)(a+b+c)^*$$

1.2 Automi

un automa a stati finiti ha un insieme di stati e un controllo che si muove da stato a stato in risposta a input esterni. Si ha una distinzione:

- **automi deterministici:** dove l'automa non può essere in più di uno stato per volta
- **automi non deterministici:** dove l'automa può trovarsi in più stati contemporaneamente

1.2.1 Automi deterministici

Un automa a stati finiti deterministico (*DFA*), un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine *deterministico* concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente. Un automa a stati finiti deterministico consiste nelle seguenti parti:

- un insieme finito di stati, spesso indicato con Q
- un insieme finito di simboli di input , spesso indicato con Σ
- una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. La funzione di transizione sarà indicata comunemente con δ. Nella rappresentazione grafica informale di automi δ è rappresentata dagli archi tra gli stati e dalle etichette sugli archi. Se q è uno stato e a è un simbolo di input, $\delta(q, a)$ è lo stato p tale che esiste un arco etichettato con a da q a p[2]
- uno stato iniziale, uno degli stati in Q
- un insieme di stati finali, o accettanti , F. L'insieme F è un sottoinsieme di Q

Nel complesso un DFA è rappresentato in maniera concisa con l'enumerazione dei suoi elementi, quindi con la quintupla:

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

con A nome del DFA, Q insiem degli stati, Δ rappresentante i simboli di input, δ la sua funzione di transizione, q_0 il suo stato iniziale e F l'insieme degli stati accettanti.
Vediamo come decidere se accettare o meno una stringa (sequenza di caratteri) in input mediante un DFA.
Ho una sequenza in input $a_1...a_n$. Parto dallo stato iniziale q_0, consultando la funzione di transizione δ, per esempio $\delta(q_0, a_1) = q_1$ e trovo lo stato in cui il DFA entra dopo aver letto a_1. Poi passo a $\delta(q_1, a_2) = q_2$ e così via, $\delta(q_{i-1}, a_i) = q_i$ fino a ottenere q_n. Se q_n è elemento di F allora $a_1...a_n$ viene accettato, altrimenti viene rifiutato.

Esempio 53. *specifico DFA che accetta tutte le strighe binarie in cui compare la sequenza 01:*

$L = \{w|w \text{ è della forma x01y, con x e y pari a 0 o 1}\} = \{01, 11010, 100011, ...\}$

o anche:

$$L = \{x01y | x, y \in \{0,1\}^*\}$$

abbiamo quindi:

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

ragioniamo sul fatto che A:

1. *se ha "già visto" 01, accetterà qualsiasi input*

2. *pur non avendo ancora visto 01, l'input più recente è stato 0, cosicché se ora vede un 1 avrà visto 01*

3. *non ha ancora visto 01, ma l'input più recente è nullo (siamo all'inizio), in tal caso A non accetta finché non vede uno 0 e subito dopo un 1*

la terza condizione rappresenta lo stato iniziale. All'inizio bisogna vedere uno 0 e poi un 1. Ma se nello stato q_0 *si vede per primo un 1 allora non abbiamo fatto alcun passo verso 01, e dunque dobbiamo permanere nello stato* q_0, $\delta(q_0, 1) = q_0$. *D'altra parte se nello stato iniziale vedo 0 siamo nella seconda condizione, uso quindi* q_2 *per questa condizione, si avrà quindi* $\delta(q_0, 0) = q_2$. *Vedo ora le transizoni di* q_2*, se vedo 0 ho che 0 è l'ultimo simbolo incontrato quindi uso nuovamente* q_2, $\delta(q_2, 0) = q_2$*, in attesa di un 1. Se arriva 1 passo allo stato accertante* q_1 *corrispondente alla prima condizione,* $\delta(q_2, 1) = q_1$. *Ora abbiamo incontrato 01 quindi può succedere qualsiasi cosa e dopo qualsiasi cosa accada potremo nuovamente aspettarci qualsiasi cosa, ovvero* $\delta(q_1, 0) = \delta(q_1, 1) = q_1$. *Si deduce quindi che:*

$$Q = \{q_0, q_1, q_2\} \text{ e } F = \{q_1\}$$

quindi:

$$A = \{\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\}\}$$

con in totale le seguenti transizioni:

$$\delta(q_0, 1) = q_0$$

$$\delta(q_0, 0) = q_2$$

$$\delta(q_2, 0) = q_2$$

$$\delta(q_2, 1) = q_1$$

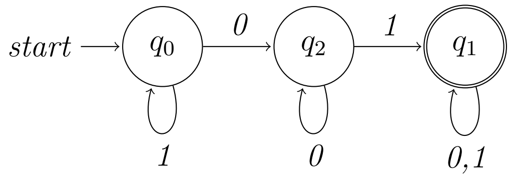
$$\delta(q_1, 0) = q_1$$

$$\delta(q_1, 1) = q_1$$

posso rappresentarle in maniera tabulare, con lo stato inizale indicato da $\rightarrow$ *e quelli accettanti con* $*$*:*

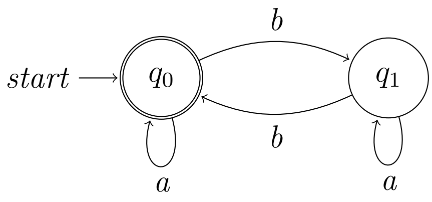
δ	*0*	*1*
$\rightarrow q_0$	q_1	q_0
$* q_1$	q_1	q_1
q_2	q_2	q_1

o col diagramma di transizione:



Esempio 54. *Trovo automa per:*

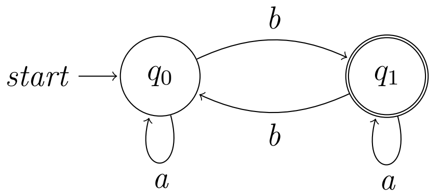
$$L = \{w \in \{a, b\}^* |\ w \text{ che contiene un numero pari di } b\}$$



ovvero se da q_0 vado a q_1 sono obbligato ab generare due b, dato che il nodo accettnate è q_0. In entrambi i nodi posso generare quante a voglio.

Esempio 55. *Trovo automa per:*

$$L = \{w \in \{a, b\}^* |\ w \text{ che contiene un numero dispari di } b\}$$



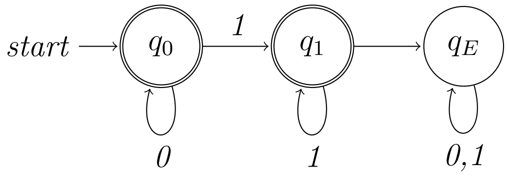
ovvero se da q_0 vado a q_1 sono obbligato ab generare una sola b, dato che il nodo accettnate è q_1. In entrambi i nodi posso generare quante a voglio e posso tornare da q_1 a q_0 per generare altre b.

Esempio 56. *Trovo automa per:*

$$L = \{w \in \{0, 1\}^* | w = 0^n 1^m\}$$

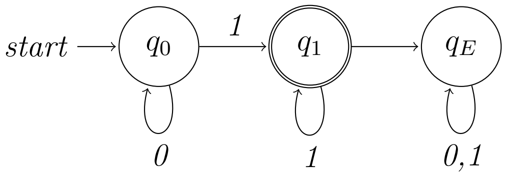
vediamo i vari casi: ***Si ha che q_E è lo stato pozzo dove vanno le stringhe venute male***

- $n, m \geq 0$:



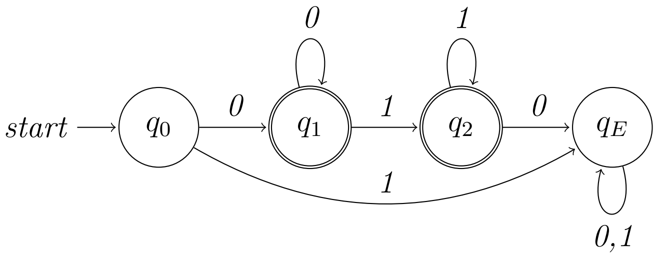
ovvero posso non generare nulla e uscire subito con q_0, generare solo un 1 e passare a q_1 e uscire oppure generare 0 e 1 a piacere con l'ultimo stato o generare 0 a piacere dal primo e 1 a piacere dal secondo.

- $n \geq 0$ $m > 0$:



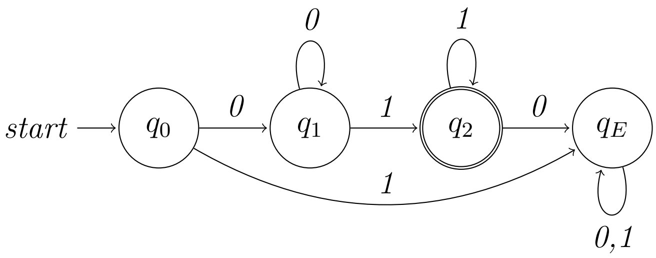
ovvero come l'esempio sopra solo che non posso uscire in q_0 in quanto almeno un 1 deve essere per forza generato

- $n > 0$ $m \geq 0$:



CHIARIRE

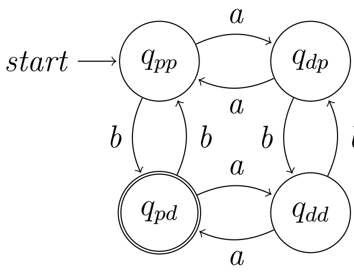
- $n, m > 0$:



CHIARIRE

Esempio 57. *Trovo automa per:*

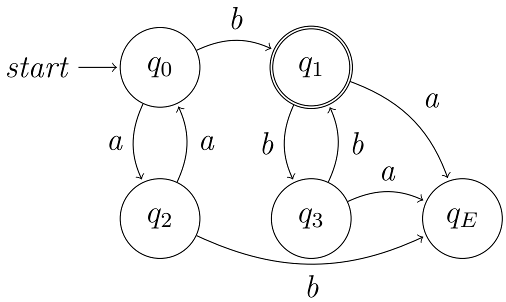
$$L = \{w \in \{a, b\}^* |\ w \text{ che contiene un numero pari di } a \text{ e dispari di } b\}$$



Esempio 58. *Trovo automa per:*

$$L = \{w \in \{a, b\}^* |\ w \text{ che contiene un numero pari di } a \text{ seguito da uno dispari di } b\}$$

$$L = \{a^{2n} b^{2k+1} | j, k \geq 0\}$$

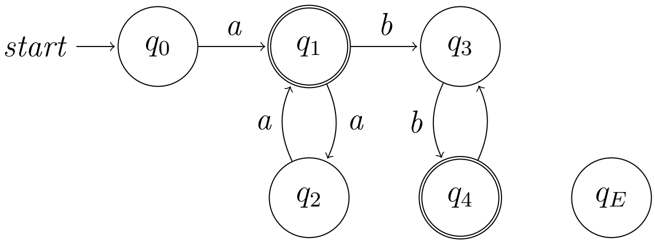


ovvero in tabella:

δ	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_E
$* q_2$	q_E	q_3
q_3	q_E	q_2
q_E	q_E	q_E

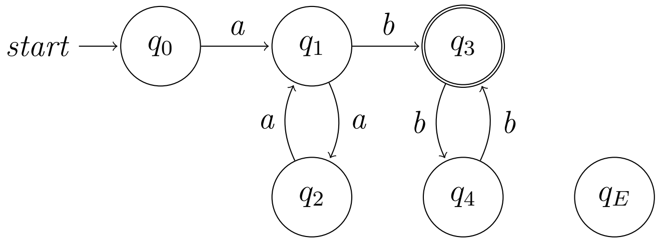
Esempio 59. *Trovo automa per:*

$$L = \{a^{2k+1} b^{2h} |\ h, k \geq 0\}$$



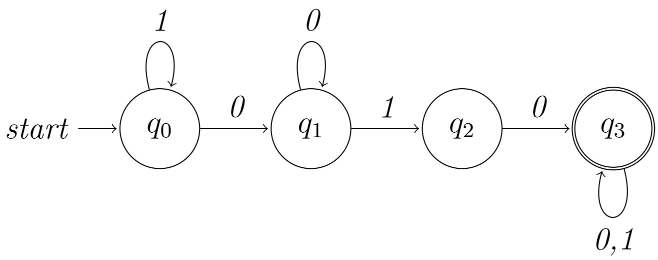
Esempio 60. *Trovo automa per:*

$$L = \{a^{2n+1}b^{2k+1} |\, n, k \geq 0\}$$



Esempio 61. *Trovo automa per:*

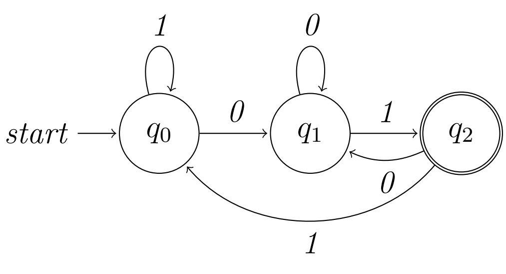
$$L = \{x010y |\, x, y \in \{o, 1\}^*\}$$



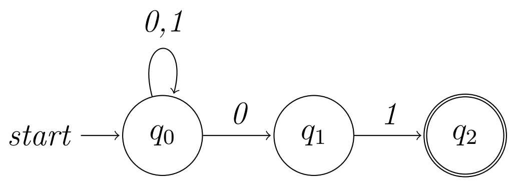
1.2.2 Automi non deterministici

Un automa a stati finiti non deterministici (*NFA*) può trovarsi in diversi stati contemporaneamente. Come i DFA accettano linguaggi regolari e spesso sono più semplici da trattare rispetto ai DFA.
Un NFA è definito come un NFA ma si ha un diverso tipo di transizione δ, che ha sempre come argomenti uno stato e un simbolo di input ma resituisce zero o più stati.

Esempio 62. *Sia* $L = \{x01 |\, x \in \{o, 1\}$ *ovvero il linguaggio formato da tutte le stringhe binarie che terminano in 01.*
Avremo il seguente automa determinsitico:



che diventa il seguente NFA:



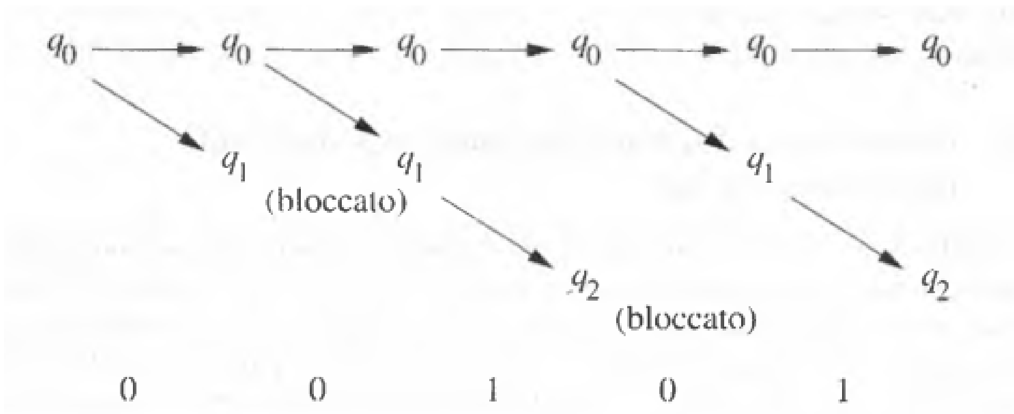
quindi con:

$$\delta(q_0, 0) = \{q_0, q_1\}$$
$$\delta(q_0, 1) = \{q_0\}$$
$$\delta(q_1, 0) = \emptyset$$
$$\delta(q_1, 1) = \{q_2\}$$
$$\delta(q_2, 0) = \emptyset$$
$$\delta(q_2, 1) = \emptyset$$

in forma tabulare:

δ	*0*	*1*
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
q_1	$\emptyset$	$\{q_2\}$
$* q_0$	$\emptyset$	$\emptyset$

vediamone la simulazione per la stringa 00101:



ovvero si parte dallo stato inziale, quando viene letto 0 si passa a q_0 e q_1, poi viene letto il secondo 0 quindi q_0 va nuovamente verso q_0 e q_1 mentre il primo q_1 muore non avendo transizioni su 0. Arriva poi l'1 quindi q_0 va solon verso q_0 e q_1 verso q_2 e sarebbe accettante ma l'input non è finito. Ora arriva 0 e q_2 si blocca mentre q_0 va sia in q_0 che in q_1. Arriva infine un 1 che manda q_0 in q_0 e q_1 in q_2 che è accettante e non avendo altri input si è dimostrata l'appartenenza della stringa al linguaggio definisco quindi un NFA come una quintupla:

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

con, a differenza dei DFA:

$$\delta : Q \times F \to 2^Q$$

Possiamo ora definire δ, delta cappuccio che prende in ingresso uno stato e l'intera stringa w. Definisco ricorsivamente:

- **caso base:** se $|w| = 0$ ovvero se $W = \varepsilon$ si ha:

$$\hat{\delta}(q, \varepsilon) = \{q\}$$

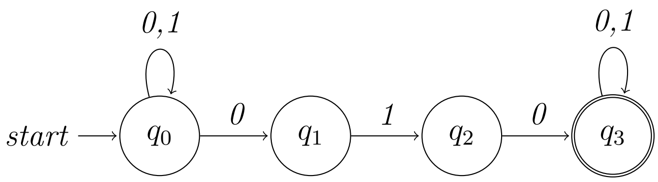
- **caso passo:** se $|w| > 0$, allora $W = xa$, $a \in \Sigma$ e $x \in \Sigma^*$. Posto $\hat{\delta}(q, x) = \{p_1, ..., p_k\}$ si ha:

$$\hat{\delta}(q, w) = cup\delta(p_i, a)$$

Per il lingauggio L accettato dall'automa si ha:

$$L(A) = \{w \in \Sigma^* | \, \hat{\delta}(q_0, q) \cap F \neq \emptyset\}$$

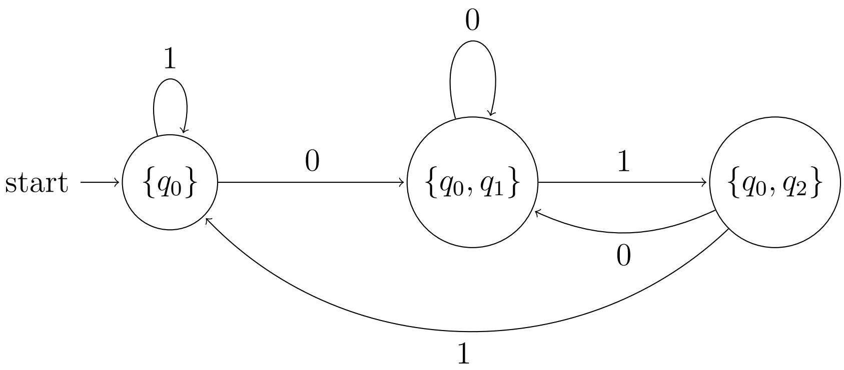
Esempio 63. *Automa per* $L = \{x010y| \, x, y \in \{0, 1\}^*\}$ *ovvero tutte le stringhe con dentro la sequenza* 010*:*



Troviamo ora un algoritmo che trasformi un NFA in un DFA.
Dall'ultimo esempio ricavo:

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\to \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$* \{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$* \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$* \{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$* \{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

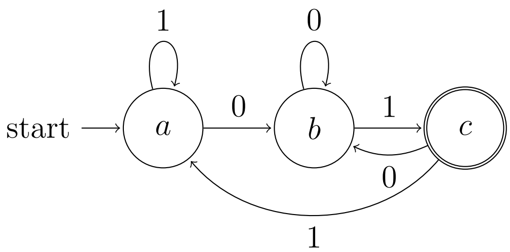
ovvero:



che è il DFA che si era anche prima ottenuto. Si hanno però dei sottoinsiemi mai raggiungibili. Si ha quindi:

	0	1
$\to \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

e definendo $\{q_0\} = a$, $\{q_0, q_1\} = b$ *e* $\{q_0, q_2\} = c$ si avrà:



Definiamo questo algoritmo che avrà:

- come input un NFA $N = (Q_n, \Sigma, \delta_N, q_0, F_N)$
- come output un DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ tale che $L(D) = L(N)$

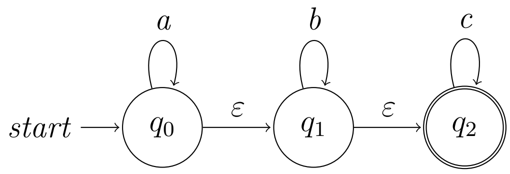
con:

- $Q_D = 2^{Q_N}$ (quindi se $Q_N = n$ si ha $|Q_D| = 2^n$)
- $F_D = \{S \subseteq Q_n | \, S \cap F_N \neq \emptyset\}$

- $\forall S \subseteq Q_N$ e $\forall a \in \Sigma$:
$$\delta_D(S, a) = \cup \delta_n(p, a)$$
per esempio:
$$\delta_D(\{q_0, q_1, q_2\}, 0) = \delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)$$

Si definisce l'$\varepsilon - NFA$, l'automa astati finiti non deterministici con ε transizioni. Con la transizione ε posso saltare i nodi, ovvero avanza senza aggiungere caratteri

Esempio 64. *Si ha* $ER = a^*b^*c^*$*, che genera:*
$$L = \{a^n b^m c^k |\, n, m, k \geq 0\}$$
si ha:



ovvero con ε *posso per esempio generare quante a voglio da* q_0 *e passare a* q_2*, uscendo senza generare altro*

Si definisce la funzione $ECLOSE : Q \to 2^Q$, con $ECLOSE(q)$ insieme degli stati raggiungibili da q tramite $\varepsilon - mosse$. Nell'esempio precedente si avrebbe:
$$ECLOSE(q_0) = \{q_0, q_1, q_2\}$$
$$ECLOSE(q_1) = \{q_1, q_2\}$$
$$ECLOSE(q_2) = \{q_2\}$$
si ha inoltre che:

- $ECLOSE 2^Q \to 2^Q \;\; P \subseteq Q$
- $ECLOSE(P) = \cup ECLOSE(p)$
- $ECLOSE(\emptyset) = \emptyset$

mettendo in tabella l'esempio precedente si ha:

	a	b	c
$* \rightarrow \{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_1, q_2\}$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$	$\{q_2\}$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

riscrivendo:

- $a = \{q_0, q_1, q_2\}$
- $b = \{q_1, q_2\}$
- $c = \{q_2\}$
- $d = \emptyset$

ovvero:

$$\delta_D(\{q_0, q_1, q_2\}, a) = ECLOSE(\delta_N(q_0, a) \cup \delta_N(q_1, a) \cup \delta_N(q_2, a))$$
$$= ECLOSE(\{q_0\} \cup \emptyset \cup \emptyset) = ECLOSE(\{q_0\})$$
$$= ECLOSE(q_0) = \{q_0, q_1, q_2\}$$

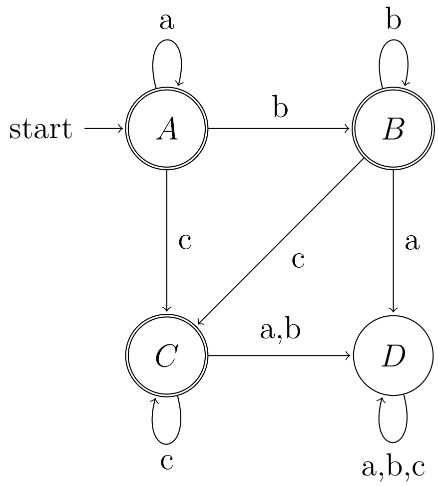
e

$$\delta_D(\{q_0, q_1, q_2\}, B) = ECLOSE(\delta_N(q_0, b) \cup \delta_N(q_1, b) \cup \delta_N(q_2, b))$$
$$= ECLOSE(\emptyset \cup \{q_1\} \cup \emptyset) = ECLOSE(\{q_1\})$$
$$= ECLOSE(q_1) = \{q_1, q_2\}$$

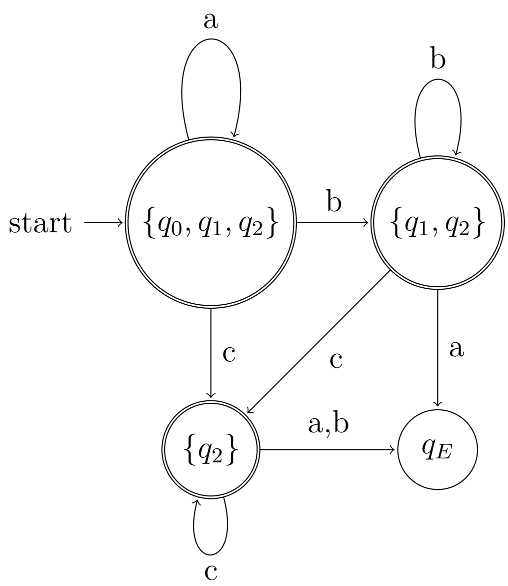
e

$$\delta_D(\{q_0, q_1, q_2\}, c) = ECLOSE(\delta_N(q_0, c) \cup \delta_N(q_1, c) \cup \delta_N(q_2, c))$$
$$= ECLOSE(\emptyset \cup \emptyset \cup \{q_2\}) = ECLOSE(\{q_1\})$$
$$= ECLOSE(q_2) = \{q_2\}$$

si ottiene quindi il seguente NFA:

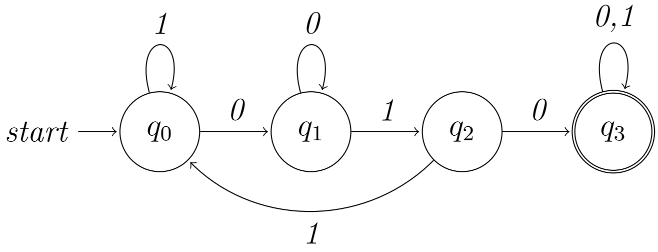


che diventa il seguente DFA:

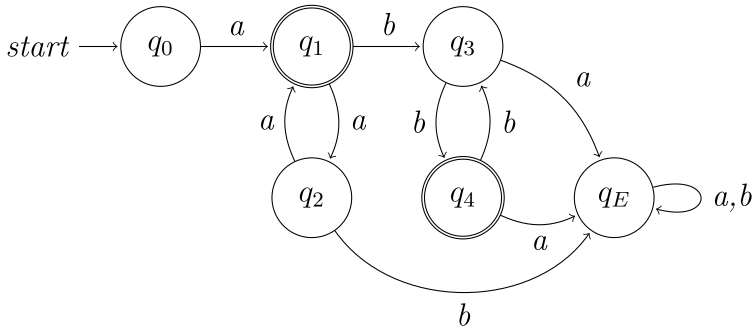


esercizi

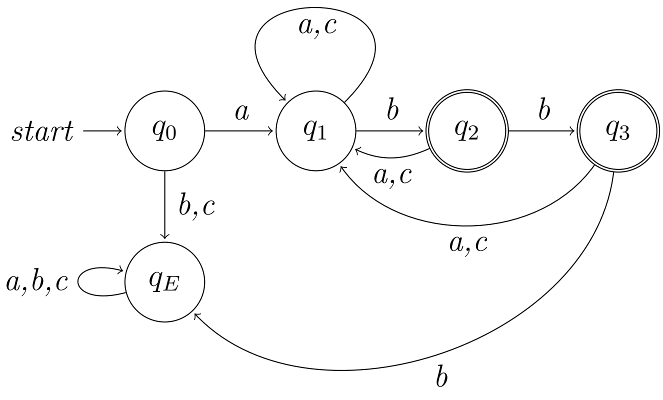
Esempio 65. *automa DFA per* $w = x010y,\ x, y \in \{0,1\}^*$ *:*
la stringa più corta è 010



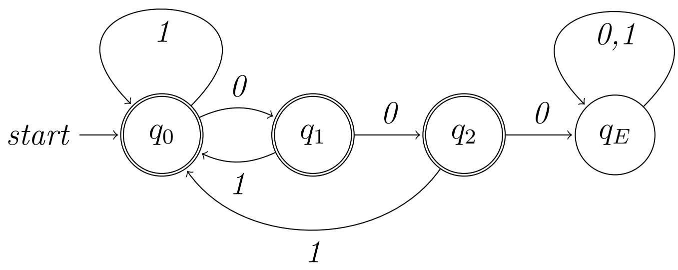
Esempio 66. *automa DFA per* $a^{2k+1}b^{2h},\ h, k \geq 0$ *:*
concatenazione di a dispari e b pari:



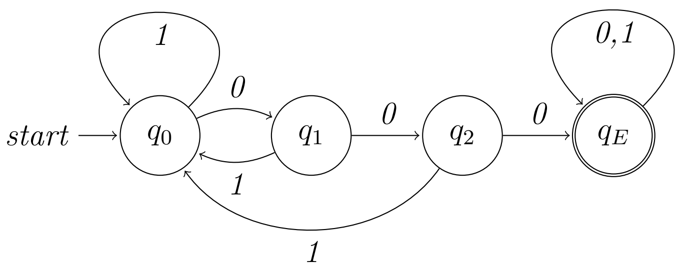
Esempio 67. *cerco DFA per stringhe inizianti con a e finenti con b, con occorrenze di b singole o a coppie, nessuna regola per c.*
per esempio abbcb è nel linguaggio



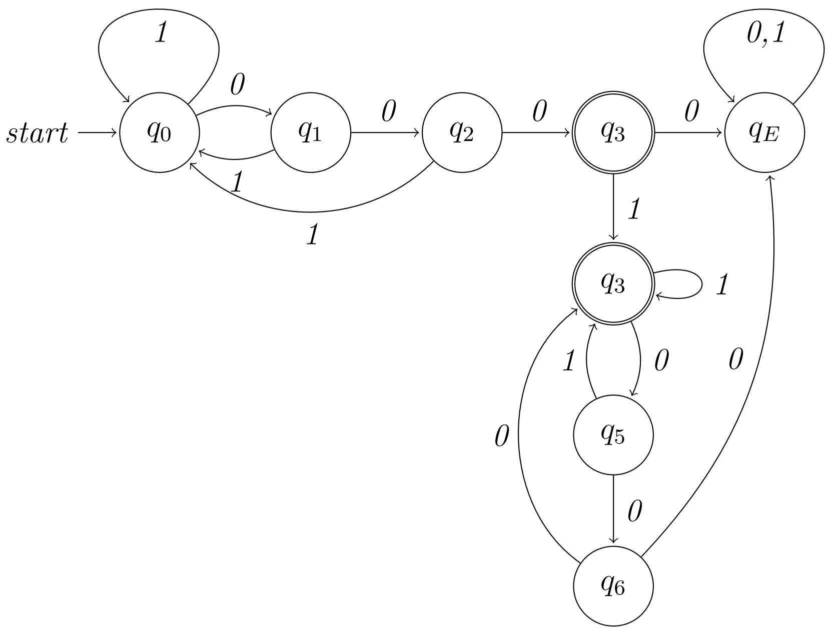
Esempio 68. *cerco DFA per stringhe di bit non contegano 000*



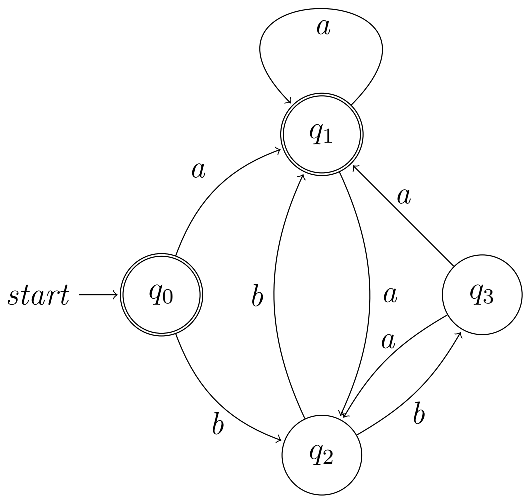
Esempio 69. *cerco DFA per stringhe di bit non contegano 000 almeno una volta*



Esempio 70. *cerco DFA per stringhe di bit che contengono 000 solo una volta*



Esempio 71. *Trasformare il seguente NFA in un DFA:*



abbiamo quindi:

$$\delta_D(\{q_0\}, a) = \delta_N(q_0, a) = \{q_1, q_2\}$$

$$\delta_D(\{q_0\}, b) = \delta_N(q_1, b) = \emptyset$$

$$\delta_D(\{q_1, q_2\}, a) = \delta_N(q_1, a) \cup \delta_N(q_2, a) = \{q_1, q_2\}cup\emptyset = \{q_1, q_2\}$$

$$\delta_D(\{q_1, q_2\}, b) = \delta_N(q_1, b) \cup \delta_N(q_2, b) = \emptyset \cup \{q_1, q_3\} = \{q_1, q_3\}$$

$$...$$

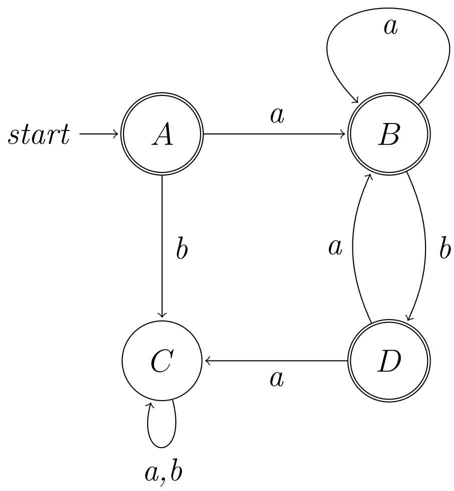
ottengo quindi:

DFA	a	b
$* \rightarrow \{q_0\}$	$\{q_1, q_2\}$	$\emptyset$
$*\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$
$*\{q_1, q_3\}$	$\{q_1, q_2\}$	$\emptyset$

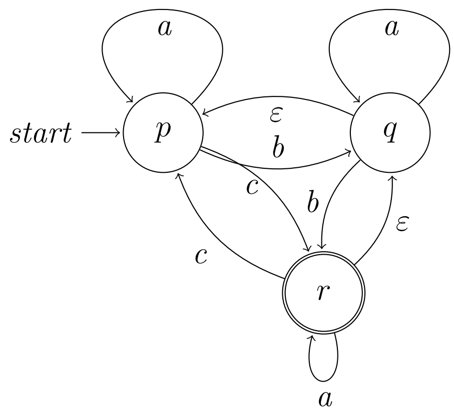
Posso ora rinominare:

- $A = \{q_0\}$
- $B = \{q_1, q_2\}$
- $C = \emptyset$
- $D = \{q_1, q_3\}$

ottengo quindi il seguente DFA:



Esempio 72. *Trasformare il seguente ε-NFA in un DFA:*



vediamo le ECLOSE:

$$ECLOSE(p) = \{p\}$$

$$ECLOSE(q) = \{p, q\}$$

$$ECLOSE(r) = \{p, q, r\}$$

si ottiene quindi:

	a	b	c
$to\,\{p\}$	$\{p\}$	$\{p, q\}$	$\{p, q, r\}$
$\{p, q\}$	$\{p, q\}$	$\{p, q, r\}$	$\{p, q, r\}$
$*\{p, q, r\}$	$\{p, q, r\}$	$\{p, q, r\}$	$\{p, q, r\}$

infatti, per esempio:

$$\delta_D(\{p\}, a) = ECLOSE(\delta_(p, a)) = ECLOSE(\{p\}) = ECLOSE(p) = \{p\}$$

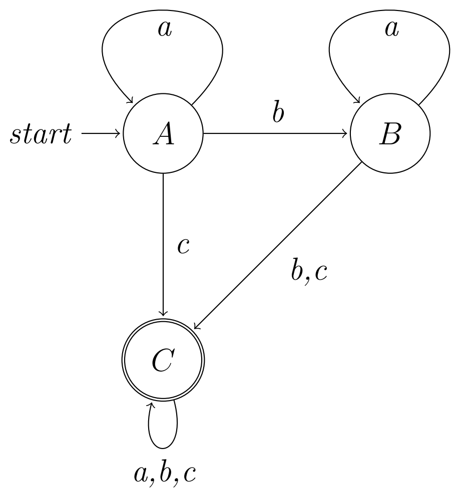
$$\delta_D(\{p, q\}, a) = ECLOSE(\delta_N(p, a) \cup \delta_N(q, a)) =$$
$$ECLOSE(\{p\} \cup \{q\}) = ECLOSE(P) \cup ECLOSE(q) = \{p\} \cup \{p, q\} = \{p, q\}$$

$$...$$

si hanno quindi le seguenti rinominazioni:

- $A = \{p\}$
- $b = \{p, q\}$
- $C = \{p, q, r\}$

ovvero:



torniamo a dare bene qualche definizione per δ in un DFA:

$$\delta\colon Q \times \Sigma^* \to Q$$

con: $q \in Q,\ \ w \in \Sigma^*\ \ e\ \ \delta(p, w) = q$

- **caso base:** $w = \varepsilon \to |w| = 0 \to \delta(q, \varepsilon) = q$
  caso passo: $|w| \neq 0 \to w = ax$ con $a \in \Sigma, x \in \Sigma^*$: $\delta(q, w) = \delta(q, ax) = \delta$ $(\delta(q, a), x)$
- **caso base:** $w = \varepsilon \to |w| = 0 \to \delta(q, w) = \delta(q, \varepsilon) = q$
  caso passo: $|w| \neq 0 \to w = xa$ con $a \in \Sigma, x \in \Sigma^*$: $\delta(q, w) = \delta(q, xa) = \delta$ $(\delta(q, x), a)$

in un NFA si ha:

- **caso base:** $\delta(q, \varepsilon = \{q\},\ \forall q \in Q$
- **caso passo:** posto $w = ax$ e $\delta(q, a) = \{p_1, ..., \grave{p}_n$ allora $\delta(q, w) = \cup\delta(p_i, x)0\{r_1, ..., r_n\}$.
  $\delta(q, a) = p$
  $\delta(q, w) = \delta(q, xa) = \delta(p, x) = r$

oppure:

- **caso base:** $\delta(q, \varepsilon) = \{q\}$, $\forall q \in Q$
- **caso passo:** posto $w = xa$ si ha $\delta(q, q) = \delta(q, xa) = \cup\delta(p, a)$ con $\delta(q, x) = \{p_1, ..., p_n\}$

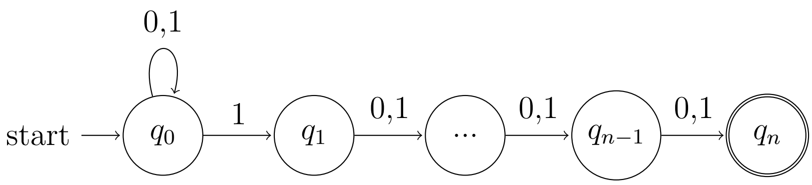
Se ho un NFA $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ con $\delta_N : Q \times \Sigma^* \to 2^{Q_N}$ che accetta un lingiaggio L posso ottenere un DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ equivalente con $Q_D = 2^{Q_N}$ e $q_{0_D} = \{q_{0_N}\}$ che accetta L.
$\forall S \subseteq Q_N$ e $\forall a \in \Sigma$ si ha:

$$F_D = \{S \subseteq Q_N \,|\, S \cap F_n \neq \emptyset\}$$

con $\delta_D(S, a) = \cup\delta_N(p, a)$ Si ha che:

$$|Q_D| = |2^{Q_N}| = 2^{|Q_N|}$$

partiamo con un esempio:



che definisce un $L \subseteq \{0,1\}^*$ tale che $w \in L$ sse n-simo elemento della fine è 1. Si ha:

$$|Q_N| = n + 1 \rightarrow |Q_D| = 2^n$$

Si ha che dato un ε-NFA $E = (Q_E, \Sigma, \delta_E, q_{0_E}, F_E)$ che riconosce L in un NFA $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ equivalenti con $Q_N = 2^{Q_E}$ stati in Q_E ε-close: *ECLOSE(S)=S* e $q_N = ECLOSE(q_E)$:

$$F_N\{S \in Q_F,\ S \subseteq Q_E | S \cap F_E \neq \emptyset\}$$

quindi:

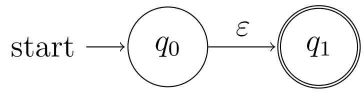
$$\forall a \in \Sigma \text{ e } \forall S = \{p_1, ..., p_k\}, \forall S \in Q_F \text{ e } Q_E$$

$$\delta_F(S, a) \text{ si ottiene } \cup\, \delta_E(p_i, a) = \{r_1, ..., r_n\}$$

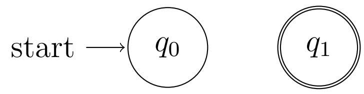
$$\delta_N(S, a) = ECLOSE(\{r_1, ..., r_n\})$$

1.2.3 Da espressioni regolari a automi E-NFA

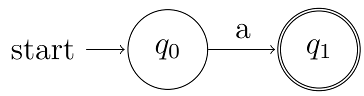
- **caso base**:
  1. se $R = \varepsilon$ ovvero $L(R) = \{\varepsilon\}$ allora:

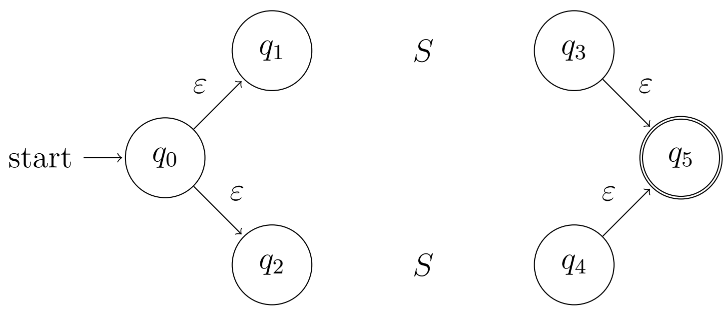
  2. se $R = \emptyset$ ovvero $L(R) = \emptyset$ allora:

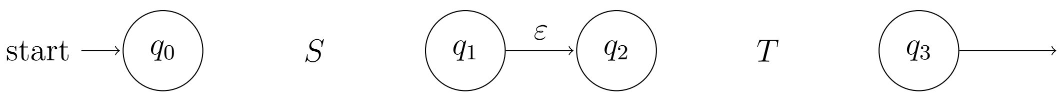
  3. se $R = a$ ovvero $L(R) = \{a\}$ allora:

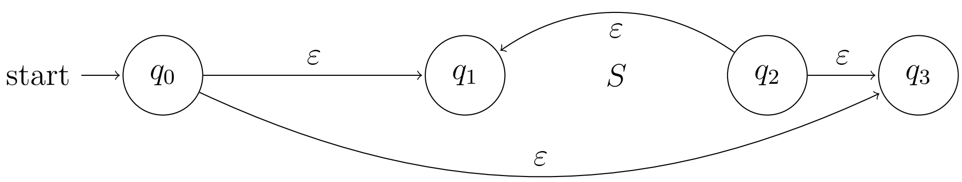
- **caso passo:**
  1. se $R = S + T$ quindi $L(R) = L(S) + L(T)$ allora:

  2. se $R = ST$ quindi $L(R) = L(S)L(T)$:

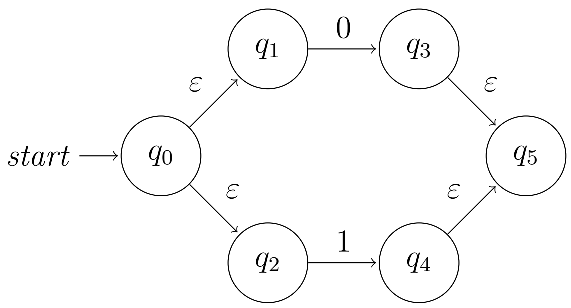
  3. se $R = S^*$ quindi $L(R) = (L(S))^*$:

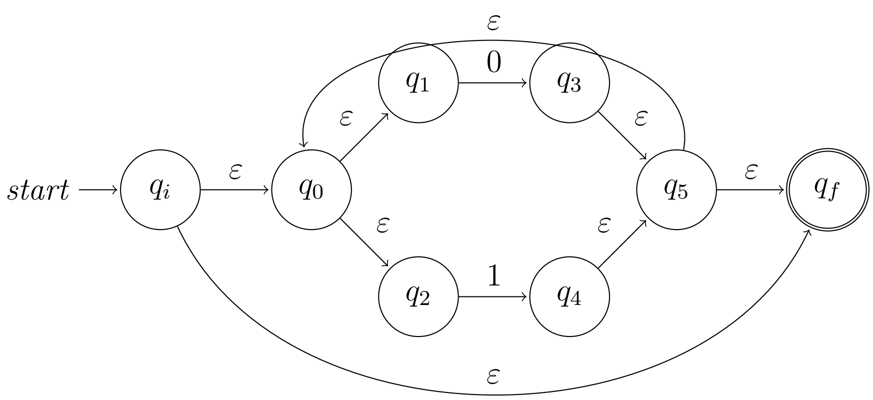
Esempio 73. *creo E-NFA per*

$$ER = (0+1)^*1(0+1)$$

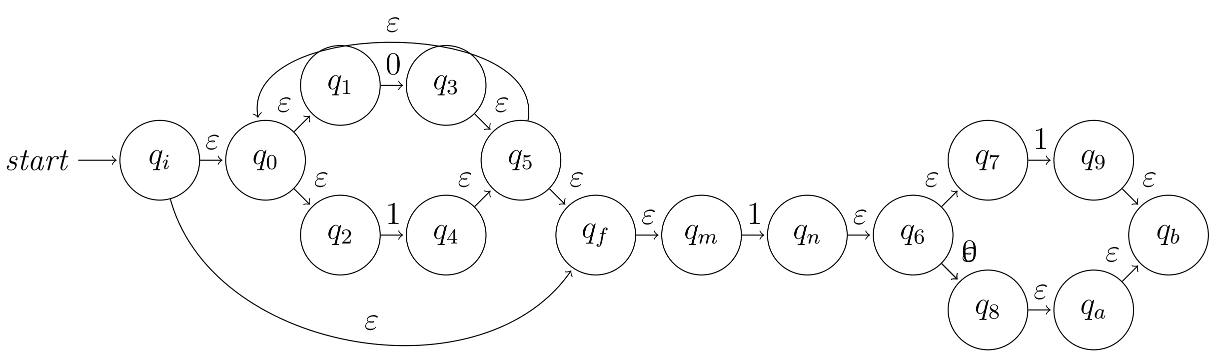
si ha che per 0+1:



da qui ottengo $(0+1)^*$



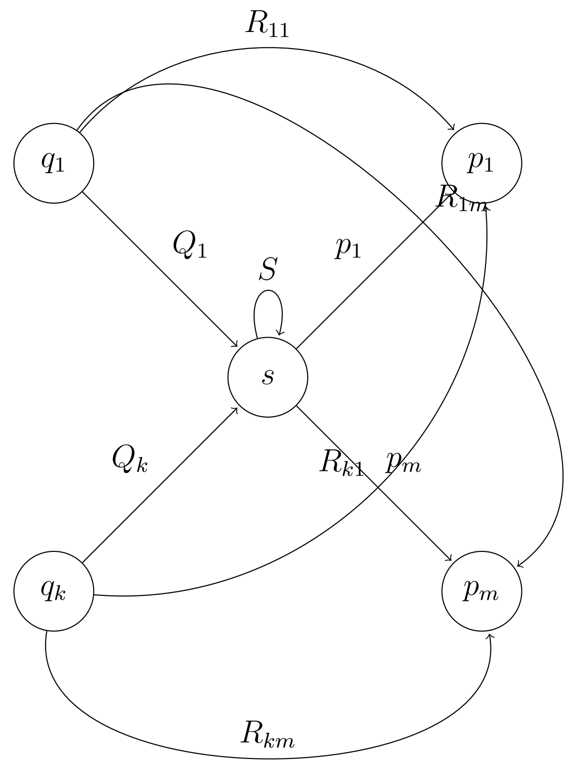
unisco e aggiungo uno nel mezzo:



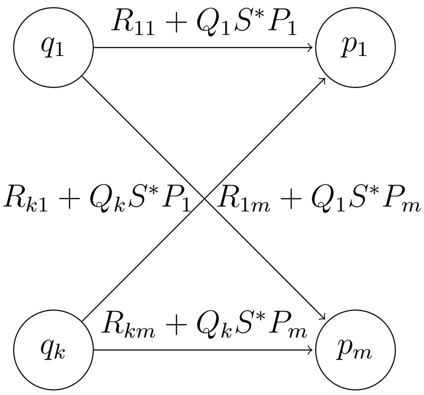
Vediamo ora l'algoritmo che trasforma DFA in una ER. Questo algoritmo permette di verificare quale linguaggio è accettato o meno dall'automa:
dato DFA $A = (Q, \Sigma, \delta, q_0, F)$ con Q e F stati non finali e Q, F, q_0 che sono i primi stati da eliminare. L'algoritmo procede per eliminazioni successive. Si costruisce quindi l'automa B che ha solo q_0 e $F = \{q_1, ..., q_k\}$. Scrivo quindi K espressioni regolari considerando solo q_0 e il k-esimo stato finale eliminando pian piano gli altri stati. Alla fine ottengo le varie espressioni regolari da unire:

$$E = E1 + E2 + \cdots + Ek$$

vediamo ora come eliminare uno stato. Partiamo dal seguente automa:

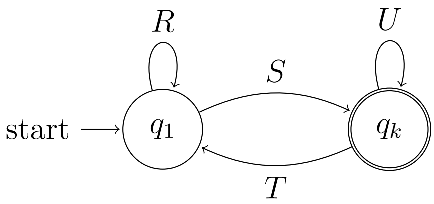


ed elimino s:



quindi quando si ha uno stato iniziale q_0 e uno finale q_1:

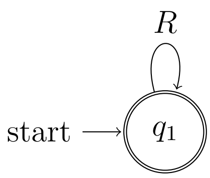
- se $q_0 \neq q_1$:



rappresenta:

$$E_i = (R + SU^*T)^*SU^*$$

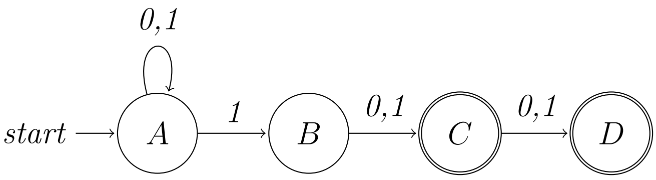
- se $q_0 = q$:



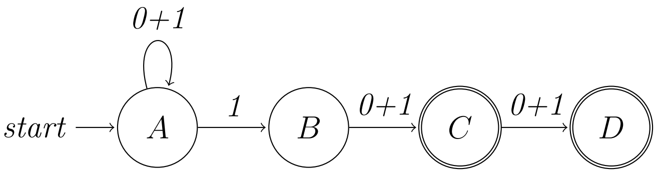
rappresenta:

$$E_i = R^*$$

Esempio 74. *passo dall'automa a l'espressione regolare:*



questo automa accetta le stringhe binarie con un uno in penultima o terzultima posizione.
Rietichetto con le ER:



elimino B che non è iniziale o finale . Il predecessore di B è A e il successore è C. Si ha quindi:

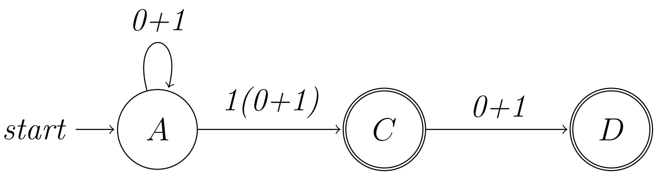
$$A \to B : 1$$

$$B \to C : 0 + 1$$

e non ho loop in B ($\emptyset$) e nessun arco tra A e C ($R_{a,c} = \emptyset$). Quindi:

$$R'_{A,C} = \emptyset + 1\emptyset^*(0 + 1) = 1(0 + 1)$$

ovvero:



Ho ora solo stati iniziali e finali, quindi $E = E_1 + E_2$. Trovo $E1$, elimino quindi C. Si ha:

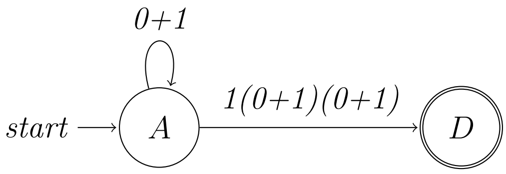
$$A \to C : 1(0 + 1)$$

$$C \to D : 0 + 1$$

e non ho loop in B ($\emptyset$) e nessun arco tra A e D($R_{A,D} = \emptyset$), ovvero:

$$R_{A,D} = \emptyset + 1(0 + 1)\emptyset^*(0 + 1) = 1(0 + 1)(0 + 1)$$

che è:



quindi:

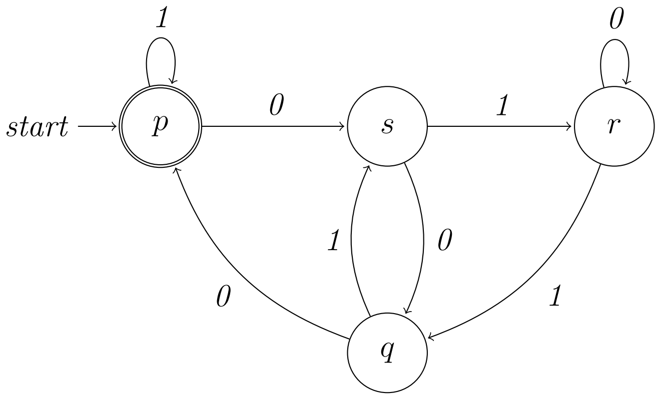
$$E_1 = ((0+1)+1(0+1)(0+1)\emptyset^*\emptyset)^*1(0+1)(0+1)\emptyset^* = (0+1)^*1(0+1)(0+1)$$

ottengo E2 eliminando D che non ha successori, quindi:

$$E_2 = (0+1)^*1(0+1)$$

e quindi infine:

$$E = E_1 + E_2 = (0+1)^*1(0+1)(0+1) + (0+1)^*1(0+1) = (0+1)^*1(0+1)(\varepsilon+0+1)$$



Esempio 75.
rimuovo r:

$$s \to r : 1$$

$$r \to q : 1$$

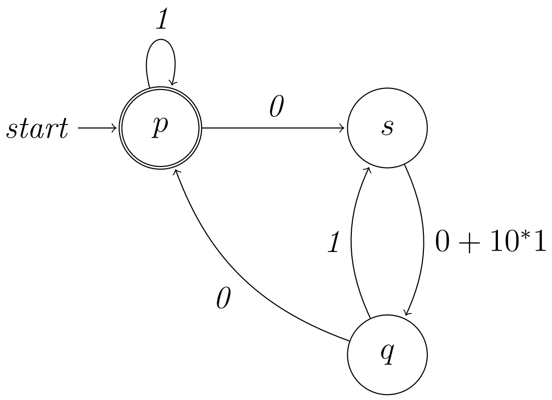
$$loop : 0$$

$$R_{s,q} = 0$$

Si ottiene quindi:

$$R^{'}_{s,q} = 0 + 10^*1$$

ovvero:



elimino ora s, che ha due predecessori, p e q, quindi:

$$p \to s : 0$$

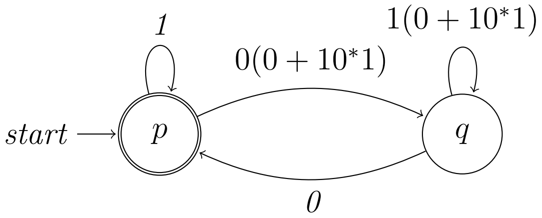
$$q \to s : 1$$

$$s \to q : 0 + 10^*1$$

non ha loop e non ha archi diretti tra p e q ($R_{p,q} = \emptyset$). Si ottengono quindi:

$$R_{p,q} = \emptyset + 0\emptyset^*(0 + 10^*1) = 0(0 + 10^*1)$$

$$R_{q,q} = \emptyset + 10^*(0 + 10^*1) = 1(0 + 10^*1)$$



elimino ora q che ha p sia come predecessore che come successore:

$$p \to q : 0(0 + 10^*1)$$

$$q \to p : 0$$

$$loop : 1(0 + 10^*1)$$

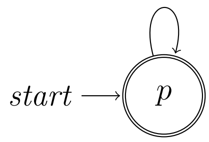
$$R_{p,p} : 1$$

ottenendo:

$$R'_{p,p} = 1 + 0(0 + 10^*1)(1(0 + 10^*1))^*0 = 1 + (00 + 010^*1)(10 + 110^*1)^*0$$

ovvero:

$$1 + (00 + 010^*1)(10 + 110^*1)^*0$$



Essendo p sia iniziale che finale si ha:

$$E = (1 + (00 + 010^*1)(10 + 110^*1)^*0)^*$$

1.2.4 Chiusura di un Linguaggio regolare

Sia REG la classe dei linguaggi regolari (ovvero ogni linguaggio regolare su un alfabeto finito non vuoto). Si ha la seguente proprietà:

$$L, M \in REG \rightarrow L \cup M \in REG$$

si può dimostrare in due maniere:

- **con le espressioni regolari:**

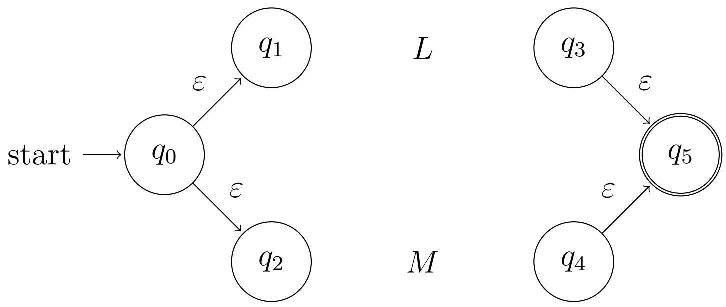
  $$L, M \in REG \rightarrow \exists R, S \text{ ER tali che } L(R) = L \text{ e } L(S) = M$$

  $$L \cup M = L(R + S) \text{ e quindi appartenente a } REG$$

- **con gli automi:**

  se $L, M \in REG \rightarrow \exists \varepsilon{-}NFA$ che con L e M va dallo stato iniziale a quello finale

  :

  accetta $L \cup M$ che quindi è REG

Inoltre siano due i due alfabeti $\Sigma \subseteq \Gamma$, si ha che:

$$\text{se L è regolare su } \Sigma \rightarrow \text{L è regolare su } \Gamma$$

inoltre:

$$\text{se L è REG su } \Sigma_1,\ \text{M è REG su } \Sigma_2,\ \text{allora } L \cup M \text{ è REG su } \Sigma_1 \cup \Sigma_2$$

Teorema 3. *Si ha che:*
Se L e M sono linguaggi regolari allora LM, ovvero la concatenazione è regolare.
Se L è un linguaggio regolare allora L^* è regolare

Teorema 4. *se $L \in REG$ su Σ allora $\overline{L} = \Sigma^* - L \in REG$*

Dimostrazione. se $L \in REG$ allora $\exists$ DFA $A = (Q, \Sigma, \delta, q_0, F)$ tale che $L(A) = L$. Costruisco $B = (Q, \Sigma, \delta, q_0, Q - F)$, $L(B) = \overline{L}$. □

si osserva che:

$$L \in REG \text{ su } \Sigma \text{ e } \Sigma \subseteq \Gamma$$

inoltre:

$$\overline{L} = \Sigma^* - L \text{ oppure } \overline{L} = \Gamma^* - L$$

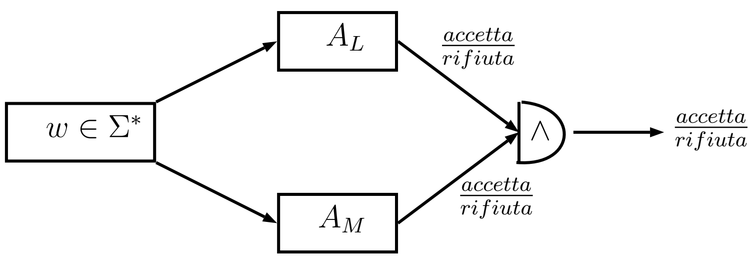
Se ho un espressione regolare per L e voglio ottenere un'espressione regolare per $\overline{L}$ devo ottenere un $\varepsilon - NFA$ che devo convertire in DFA. A quel punto complemento con F e ottengo un'espressione regolare per $\overline{L}$.
Passiamo ora all'intersezione:

Teorema 5. *se L e M sono regolari allora la loro intersezione è regolare*

Dimostrazione.

$$L, M \in REG \rightarrow \exists A_L, A_M (DFA) \text{ tali che } L(A_L) = L \text{ e } L(A_M) = M$$

e quindi:



□

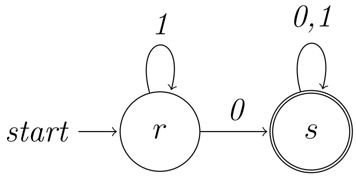
Passiamo al problema della **chiusura dei linguaggi regolari rispetto all'intersezione**, ovvero che due automi che usano entrambi contemporaneamente una certa stringa in input, il ché non è possibile. Si risolve usando l'**automa prodotto**:

$$A = (Q_L \times Q_M, \Sigma, \delta, (q_{OL}, q_{OM}).F_L \times F_M)$$

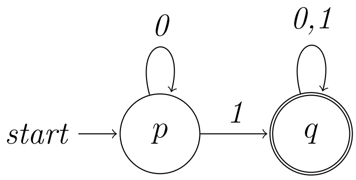
con:

$$\delta(p, q), a) = (\delta_L(p, a), \delta_M(q, a))$$
$$A_L = (Q_L, \Sigma, \delta_L, q_{OL}, F_L)$$
$$A_M = (Q_M, \Sigma, \delta_M, q_{OM}, F_M)$$

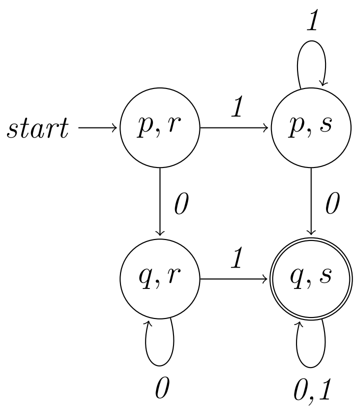
Esempio 76. *Siano:*
A_L*, stringhe binarie con almeno uno zero:*



A_M*, stringhe binarie con almeno un uno:*

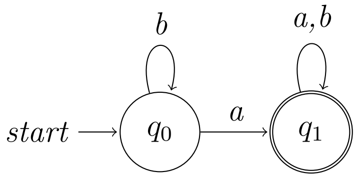


ottengo l'automa prodotto:

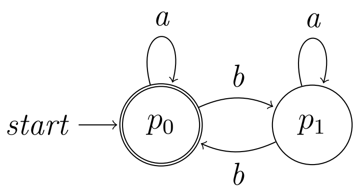


$$\delta((p,r),0) = (\delta_L(p,o), \delta_M 8r, o)) = (q,r)$$

Esempio 77. *Siano:*
A, stringhe contenenti a e b contenenti almeno una b:



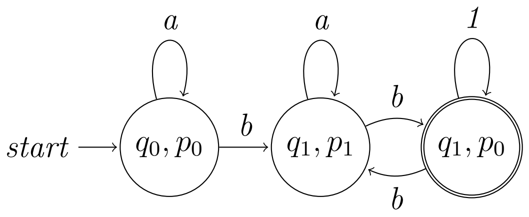
A_M, stringhe contenenti a e b contenenti $2n$, $n \geq 0$ b:



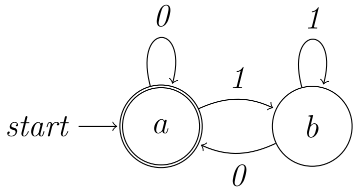
si ha che $L(A \wedge B)$ è il linguaggio formato da stringhe contenenti a e b contenenti $2n$, $n \geq 1$ b: faccio la tabella:

	a	b
$\rightarrow (q_0, p_0)$	(q_0, p_0)	(q_1, p_1)
(q_1, p_1)	(q_1, p_1)	(q_1, p_0)
$*(q_1, p_0)$	(q_1, p_0)	(q_1, p_1)

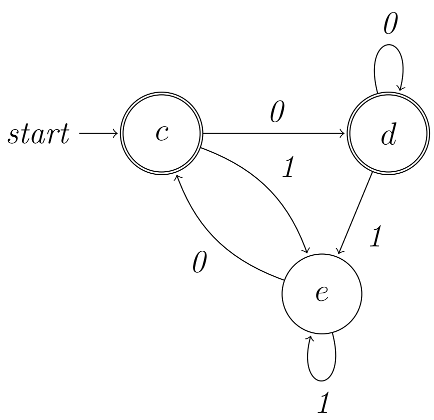
ottengo l'automa prodotto:



Esempio 78. *si hanno due linguaggi $L(A)$ e $L(B)$ tali per cui la loro intersezione non è vuota.*
Sia A:



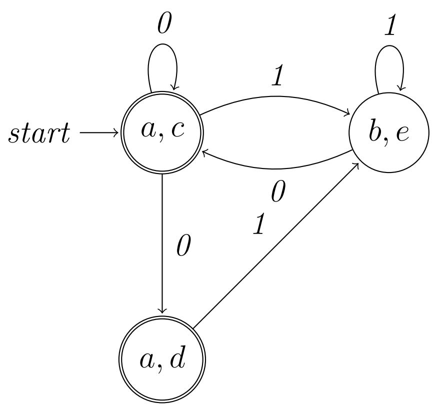
e B:



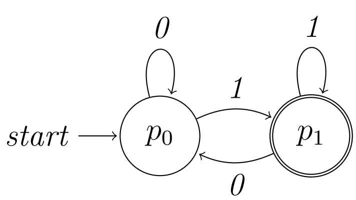
faccio la tabella:

	0	1
$\stackrel{*}{\rightarrow}(a,c)$	(a,d)	(b,e)
$*(a,d)$	(a,d)	(b,e)
(b,e)	(a,c)	(b,e)

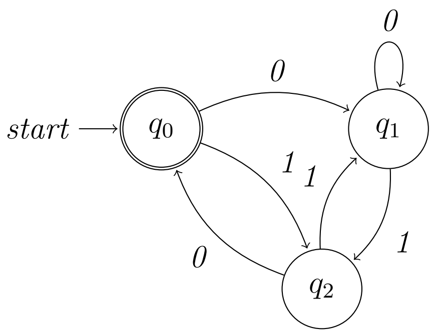
e ottengo:



Esempio 79. *Sia A:*



e B:



faccio la tabella:

	0	1
$\rightarrow (p_0, q_0)$	(p_0, q_1)	(p_1, q_2)
(p_0, q_1)	(p_0, q_1)	(p_1, q_2)
(p_1, q_2)	(p_0, q_0)	(p_1, q_1)
(p_1, q_1)	(p_0, q_1)	(p_1, q_2)

non si raggiungono stati finali quindi $L(A \cap B) = \emptyset$

1.2.5 Minimizzazione

definiamo la **relazione di equivalenza tra stati**:

Definizione 9. *Sia* A *un* DFA, $A = ()Q, \Sigma, \delta, q_0, F)$. *Siano:*

$$p, q \in Q \rightarrow p \approx q \text{ vale se } \forall w \in \Sigma^* \;\; \hat{\delta}(p, w) \in F \longleftrightarrow \hat{\delta}(q, w) \in F$$

inoltre p *e* q *sono distinguibili se* $p \not\approx q$ *ovvero:*

$$\exists w \in \Sigma^* \text{ tale che:}$$

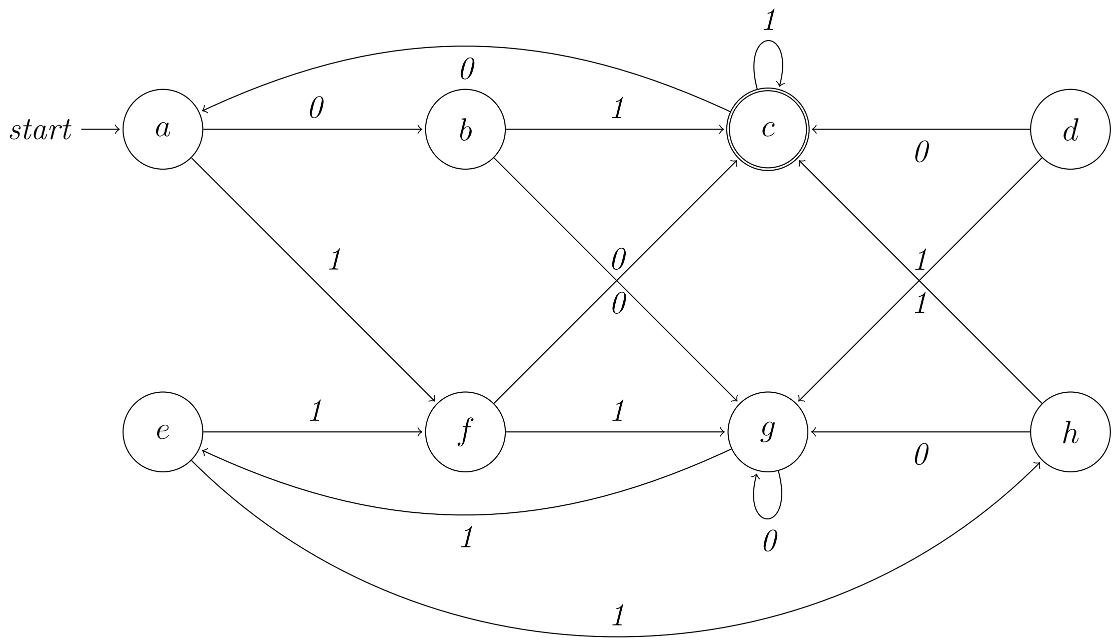
$$\hat{\delta}(p, w) \in F \text{ e } \hat{\delta}(q, w) \notin F \text{ o } \hat{\delta}(p, w) \notin F \text{ e } \hat{\delta}(q, w) \in F$$

quindi si ha una **relazione di equivalenza** e quindi si hanno le tre proprietà:

- **riflessività:** $\forall p \in Q, p \approx p$
- **simmetricità:** $\forall p, q \in Q, p \approx q \rightarrow q \approx p$
- **transitività:** $\forall p, q, r \in Q, p \approx q \vee q \approx r \rightarrow p \approx r$

e quindi **in ogni classe di equivalenza ci sono stati tra loro equivalenti** due stati, uno finale e uno non finale, sono sicuramente distinguibili, basti pensare alla stringa vuota.

Esempio 80. *Analizziamo il seguente automa:*



cerco di capire se a e g sono equivalenti:

- *ε non li distingue perché entrambi sono non accettanti*
- *0 non li distingue perché li porta rispettivamente in b e g, che non sono accettanti*
- *1 non li distingue perché li porta rispettivamente in f e e, che non sono accettanti*
- *01 li distingue perché $\hat{\delta}(a, 01) = c$ e $\hat{\delta}(g, 01) = e$ e il primo è accettante mentre il secondo no, quindi i due stati non sono equivalenti*

si verifica che invece a e e sono equivalenti

diamo una definizione ricorsiva dell'equivalenza tra stati:

- **caso base:** se $p \in F$ e $q \notin F$ o viceversa si ha che gli stati sono distinguibili
- **caso passo:** se per $a \in \Sigma$ gli stati $r = \delta(p, a)$ e $s = \delta(q, a)$ sono distinguibili allora anche p e q lo sono

quindi se un certo arco w porta due stati a due diversi di cui uno è accettante essi sono distinti.
Esiste anche un algoritmo detto **riempi-tabella**:
partiamo dalla seguente tabella

b		/	/	/	/	/	/
c			/	/	/	/	/
d				/	/	/	/
e					/	/	/
f						/	/
g							/
h							
	a	b	c	d	e	f	g

c è accettante quindi segniamo tutte le caselle relative a c per il caso base:

b		/	/	/	/	/	/
c	x	x	/	/	/	/	/
d			x	/	/	/	/
e			x		/	/	/
f			x			/	/
g			x				/
h			x				
	a	b	c	d	e	f	g

poiché, per esempio c, h è distinguibile e gli stati e e f con 0 vanno in h e c si ha che anche e e f sono distinguibili, segno quindi nelle caselle appropriate.
Proseguo controllando anche le altre coppie e riempio le caselle. Ottengo:

b	x	/	/	/	/	/	/
c	x	x	/	/	/	/	/
d	x	x	x	/	/	/	/
e		x	x	x	/	/	/
f	x		x	x	x	/	/
g	x	x	x	x	x	x	/
h	x		x	x	x	x	x
	a	b	c	d	e	f	g

quindi si ha solo che:

$$a \approx e \; b \approx h \; d \approx f$$

si ha che:

Teorema 6. *due stati non distinti dall'algoritmo riempi-tabella sono equivalenti. Se ne calcola la complessità:*

$$|Q| = n \to \frac{n(n-1)}{2} \ \textit{caselle} = O(n^2)$$

se ho una crocetta a iterazione ho il caso peggiore $n^2 n^2 = O(n^4)$

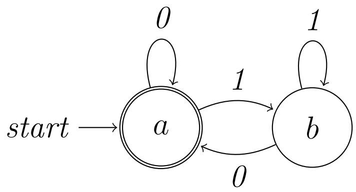
quindi per vedere se due linguaggi regolari sono equivalenti si ha che:

$$L, M \in REG$$

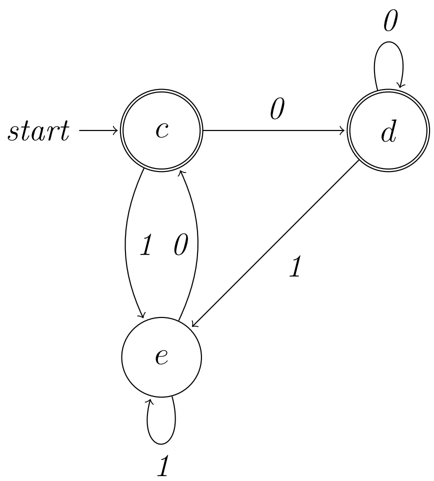
$$\exists A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$
$$\exists A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

costruisco $A = (Q_L \cup Q_M, \Sigma, \delta, q_L, F_L \cup F_M)$
δ è δ_L per Q_L e δ è δ_M per Q_M per vedere se $q_L \approx q_M$ uso il riempi tabella e se sono equivalenti si ha che $L = M$

Esempio 81. *Sia A:*



e B:



entrambi i linguaggi accettano $(\varepsilon + (0+1)^*0)$ *applico la tabella:*

b	x	/	/	/
c		x	/	/
d		x		/
e	x		x	x
	a	b	c	d

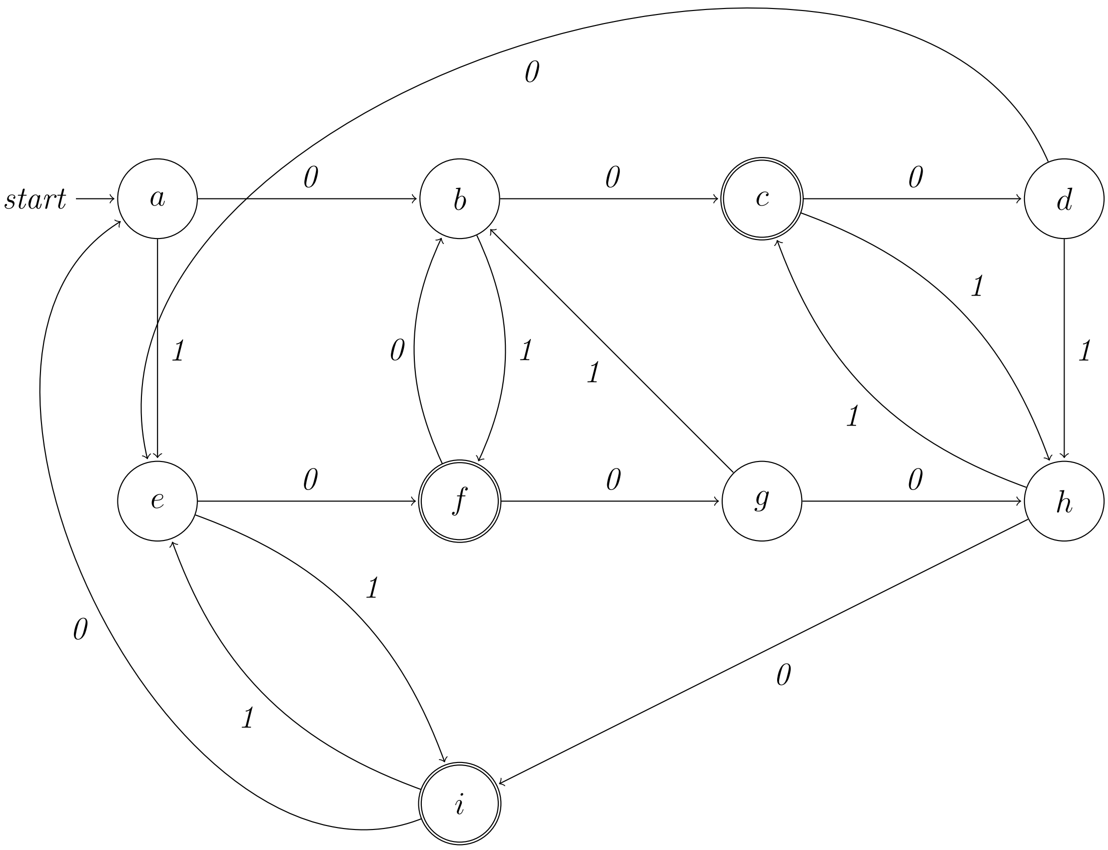
e quindi dato che $a \approx c$ si ha che i due linguaggi sono lo stesso linguaggio. ***CHIEDERE SPIEGAZIONI A RIGUARDO***

passiamo ora alla **minimizzazione**, che prende in input un DFA A e restituisce un $DFA\ A_{min}$ tale che $L(A) = L(A_{min})$ e che A_{min} ha il numero più piccolo possibile di stati per distinguere $L(A)$.
Procedo così:

1. si rimuovono gli stati non raggiungibili
2. applico la tabella per scoprire le tabelle di equivalenza, gli stati del nuovo automa sono le classi di equivalenza

si ha che lo stato iniziale è la classe di equivalenza dello stato finale e gli stati finali sono le classi di equivalenza degli stati finali.
La minimizzazione si dimostra per assurdo:
Sia M il DFA ottenuto dalla tabella. Suppongo esista un DFA N tale che $L(A) = L(N)\,|Q_N| < |Q_M|$. I due stati iniziali sono indistinguibili. Sia $p \in M$ $q \in N$ tali $p \approx q$ $\forall a \in \Sigma$ quindi $\delta p, a) = \delta(q, a)$, Ogni stato $p \in Q_;$è quindi indistinguibile da almeno uno stato di N. Avendo però N meno stati si avranno almeno due stati di M indistinguibili dallo stesso di N ma non sono indistinguibili per la tabella. SI ha un assurdo.

Esempio 82. *Si ha il seguente automa:*



faccio la tabella

b		/	/	/	/	/	/	/
c	x	x	/	/	/	/	/	/
d		z	x	/	/	/	/	/
e	x		x	x	/	/	/	/
f	x	x		x	x	/	/	/
g		x	x		x	x	/	/
h	x		x	x		x	x	/
i	x	x		x	x		x	x
	a	b	c	d	e	f	g	h

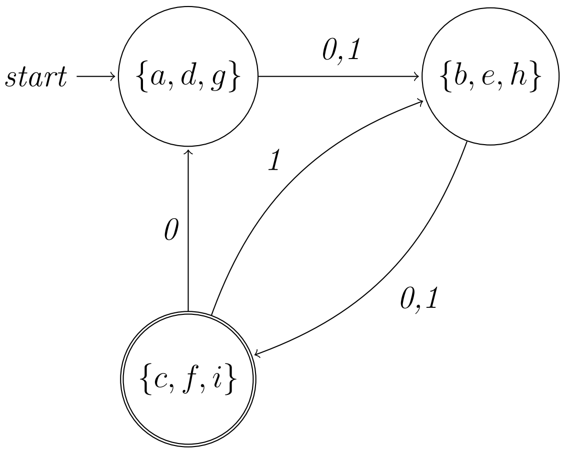
al caso base segno tutte le caselle degli stati accettanti tranne quelle che incrociano con altri stati accettanti e poi completo il resto della tabella. Ottengo:

$$a \approx d \approx g$$

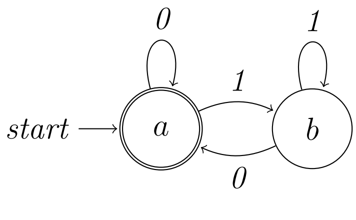
$$b \approx e \approx h$$

$$c \approx f \approx i$$

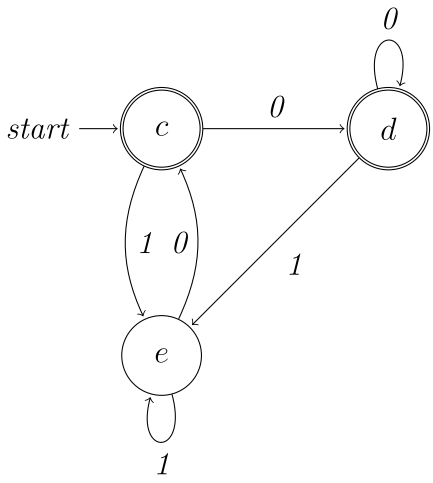
e quindi ottengo, l'automa minimizzato:



Esempio 83. *Sia A:*



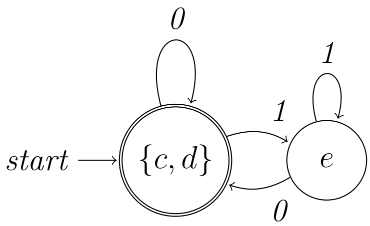
e B:



riduco B:

d		/
e	x	x
	c	d

quindi $c \approx d$. *Ottengo quindi:*



quindi A è B minimizzato, per questo riconoscevano lo stesso linguaggio

La tabella non funziona ovviamente por gli NFA

1.2.6 Pumping Lemma per i linguaggi regolari

Questo lemma serve a dimostrare che un linguaggio L non è regolare. Si procede per assurdo.
Partiamo da un esempio:

$$L_{01} = \{0^n 1^n |\, n \geq 1\} = \{01, 0011, ...\}$$

supponiamo che questo linguaggio sia rappresentabile da un DFA A con k stati ($L(A) = L_{01}$).

0^k implica $k+1$ prefissi: $\varepsilon, 0, 00, 000, ..., 0^k$ e quindi $\exists i, j$ tali che 0^i e 0^j finiscono nello stesso stato. Allora l'automa è ingannabile e accetterebbe 0^i0^j con $i \neq j$.
Formalmente si ha:

Teorema 7. *Sia $L \in REG$ allora $\exists n$ che dipende da L tale che $\forall w \in L$ con $|w| \geq n$, w può essere scomposta in tre stringhe $w = xyz$ in modo che:*

$$y \neq \varepsilon$$

$$|xy| \leq n$$

$$\forall k \geq 0 \;\; zy^kz \in L$$

in altre parole posso sempre trovare una stringa non vuota y non troppo distante dall'inizio di w, da replicare da ripetere o cancellare ($k = 0$) senza uscire dal linguaggio L

Dimostrazione. essendo $L \in REG$ esiste un DFA A tale che $L = L(A)$. Suppongo $|Q| = n$ e considero $w = a_1a_2...a_n$ con $m \geq n$. Sia:

$$p_i = \hat{\delta}\,(q_0, a_1, ...a_i) \;\forall i = 0, 1, ..., n$$

quindi:

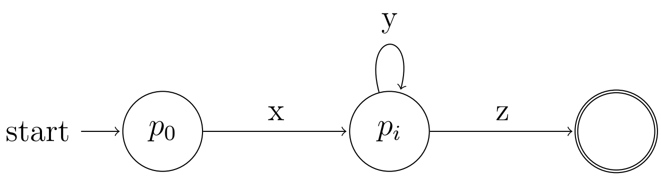
$$p_0 = q_0, p_1, ..., p_n \rightarrow n + a \textit{ stati}$$

allora $\exists i, j$, con $0 \leq i < j \leq n$ tali che $p_i = p_j$. Scompongo ora w in $w = xyz$ con:

$$x = a_1a_2...a_n$$

$$y = a_{i+1}a_{i+2}...a_j$$

$$z = a_{j+1}a_{j+2}...a_m$$



□

Esempio 84. *mostriamo che L01 non è regolare. Suppongo per assurdo che sia regolare, allora vale il pumping lemma. Sia $n \in \mathbb{N}$ la costante del pumping lemma. Pongo $w = 0^n1^n = xyz$ tale che $|xy| = n$, ovvero $|xy| = 0^n$, quindi xy è formato da soli 0. Poniamo $x = 0^{n-1}$ $y = 0 \neq \varepsilon$ $z = 1^n$. Però per il pumping lemma $\forall k \geq 0 \;\; xy^kz \in L_{01}$.*

$$xy^kz = 0^{n-1}0^k1^n = 0^{n+k-1}1^n \in L_{01}$$

ma se $k \neq 1$ ciò non è vero e quindi il linguaggio non è regolare

Esempio 85. *$L = \{w \in \{0,1\}^* |\, w = w^R\}$, linguaggio delle stringhe palindrome, è regolare?*
suppongo per assurdo di sì, con n costante del pumping lemma.
$w = 0^n 1 0^n$ quindi $x = 0^{n-1}$ $y = 0$ $z = 10^n$. Per $k = 0$ si ha $xy^kz = 0^{n-1}10^n \notin L$, quindi non è regolare

Esempio 86. *$L = \{1^p |\, p\ primo\} = \{11, 111, 11111\}$ è regolare?*
$w = 1^p = 1^{n-1}11^{p-n}$ con:

$$x = 1^{n-1}\ y = 1\ z = 1^{p-n}$$

per $k = 0$ si ha 1^{p-1} e $p-1$ non è , quindi non è regolare

Esempio 87. *$L = \{0^n1^m |\, n \leq m\}$ è regolare?*
$w = 0^{n-l}0^l1^m$ con:

$$x = 0^{n-l}\ y = 0^l\ z = 1^m$$

con $|y| = l$ e $0 < l \leq n$
quindi $\forall k \geq 0$ si ha $xy^kz \in L = 0^{n-l}0^{kl}1^m$ con:

$$x = 0^{n-l}\ y = 0^{kl}\ z = 1^m$$

scelgo k tale che $b - l + kl = n + (k-1)l > m \to k > 1 + \frac{m-n}{l}$ quindi non è regolare

Esempio 88. *$L = \{0^n1^m |\, n \geq m\}$ è regolare?*
$w = 0^{n-l}0^l1^m$ con:

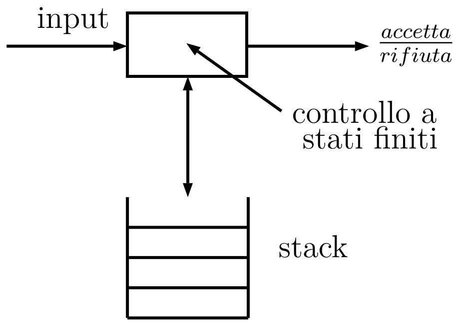
$$x = 0^{n-l}\ y = 0^l\ z = 1^m$$

con $|zy| \leq n$ $y \neq \varepsilon$ $|y| = l$ e $0 < l \leq n$
scelgo l tale che $n - l < m$ quindi non è regolare

su appunti dispense altri esempi

1.3 Automi a Pila

Si introduce un nuovo tipo di automa, il PDA (push down automata) che può essere pensato come un $\varepsilon - NFA$ col supporto di una pila (stack):



e viene definito un PDA P come:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$$

con;

- Q: insieme finito e non vuoto di stati
- Σ: alfabeto di simboli di input
- Γ: alfabeto di simboli di stack
- $q_0 \in Q$: stato iniziale
- $z_0 \in \Gamma \backslash \Sigma$: simbolo iniziale dello stack
- $F \in Q$: insieme degli stati accettanti o finali

si ha che:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\} \times \Gamma \to 2^{Q \times \Gamma^*}$$

quindi:

$$\delta(q_0, a, X) = \{(p_1, X_1), (p_2, X_2), ...\}\, insieme\ \ finito\ p_i \in Q\ X_i \in \Gamma^*$$

si hanno dei casi particolari:

- lo stato p potrebbe coincidere con Q e si avrebbe un cappio
- se $\Gamma = \varepsilon$ si ha il pop di X dallo stack
- se $\Gamma = X$ si lascia lo stack invariato
- se $\Gamma = Y \neq X$ si ha la sostituzione di X con Y in cima allo stack
- se Γ è una stringa di simboli si ha il la rimozione di X dallo stack e l'aggiunta a uno a uno dei simboli nello stack

Esempio 89. *Trovo PDA per il linguaggio delle stringhe binarie palindrome di lunghezza pari:* $L = \{ww^R | w \in \{0,1\}^*\}$. *Con R che indica rovesciato*
Si ha la CFG $G = (\{P\}, \{0,1\}, Prod, P)$ *tale che:*

$$P \to 0P0|1P1|\varepsilon$$

si hanno quindi tre stati:

- q_0 *che è quello iniziale che legge* w *e spinge i dati sullo stack*
- q_1 *che letta* w *legge i simboli di* wR *e li confronta con quelli dello stack*
- q_2 *sarà la stringa accettata*

descriviamo formalmente l'automa con la funzione di transizione δ. *PDA* $P = (\{q_0, q_1, q_2\}, \{0,1\}, \{o, 1, z_0\}, \delta, q_0, z_0, \{q_2\})$ *ovvero:*

$$\delta(q_0, 0, z_0) = \{(q_0, 1z_0)\}$$

$$\delta(q_0, 1, z_0) = \{(q_0, 0z_0)\}$$

$$\delta(q_0, 0, 0) = \{(q_0, 00)\}$$

$$\delta(q_0, 0, 1) = \{(q_0, 01)\}$$

$$\delta(q_0, 1, 0) = \{(q_0, 10)\}$$

$$\delta(q_0, 1, 1) = \{(q_0, 11)\}$$

$$\delta(q_0, \varepsilon, z_0) = \{(q_0, z_0)\}$$

$$\delta(q_0, \varepsilon, 0) = \{(q_0, 0)\}$$

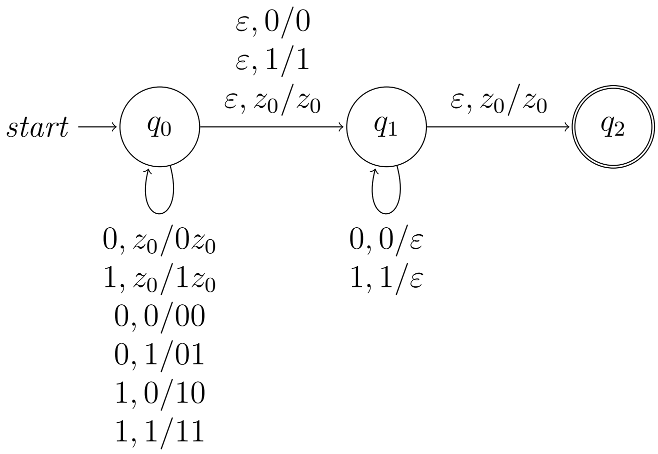
$$\delta(q_0, \varepsilon, 1) = \{(q_0, 1)\}$$

$$\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$$

$$\delta(q_2, 1, 1) = \{(q_1, \varepsilon)\}$$

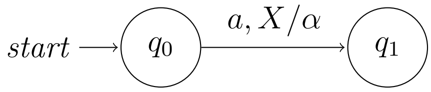
$$\delta(q_2, \varepsilon, z_0) = \{(q_2, 0z_0)\}$$

otteniamo il seguente PDA:



e si definisce questa notazione per gli archi:

$$(p, \alpha) \in \delta(q, a, X)$$



analizziamo meglio i PDA. Si ha che la **descrizione istantanea (ID)** di un PDA è una tripla:

$$ID : (q, w, \gamma)$$

con $q \in Q$ stato attuale $w \in \Sigma^*$ input rimanente e $\gamma \in \Gamma^*$ contenuto attuale dello stack.
Definiamo ora il concetto di **mossa in un passo** dato $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ la mossa è una relazione $\overset{\vdash}{p}$:

$$(p, \alpha) \in \delta(q, a, X) \ allora \ \forall w \in \Sigma^* \ e \ \forall \beta \in \Gamma^* \rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta$$

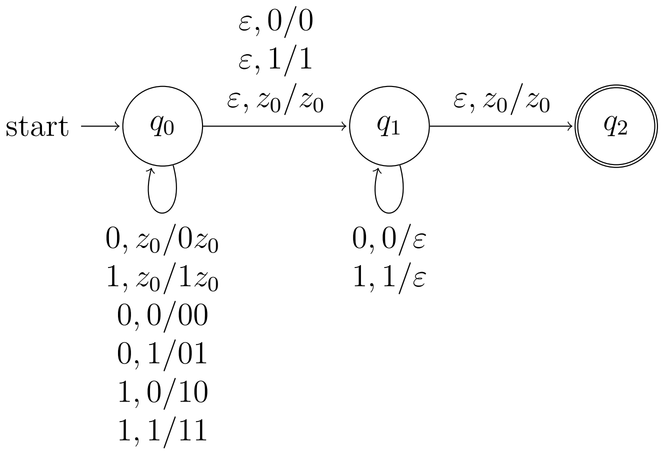
e

$$(p, \alpha) \in \delta(q, \varepsilon, X) \ allora \ \forall w \in \Sigma^* \ e \ \forall \beta \in \Gamma^* \rightarrow (q, w, X\beta) \vdash (p, w, \alpha\beta$$

ora possiamo anche definire la relazione con 0 o più mosse in forma induttiva $\overset{*}{\underset{P}{\vdash}}$:

- **caso base:** $\forall ID\ I, I \overset{*}{\vdash} I$
- **caso passo:** $I \overset{*}{\vdash} J$ se $\exists ID\ K$ tale che $I \vdash K$ e $K \overset{*}{\vdash} J$

vediamo un esempio con un PDA che accetta $ww^R|\, w \in \{0,1\}^*$:



e prendiamo la stringa 1111:

$q_0, 1111, z_0$

$q_0, 111, 1z_0$ $\quad (q_1, 1111, z_0) \to (q_2, 1111, z_0) \to X$

$q_0, 11, 11z_0$ $\quad (q_1, 111, 1z_0) \to (q_1, 11, z_0) \to (q_2, 11, z_0) \to X$

$q_0, 1, 111z_0$ $\quad (q_1, 11, 11z_0) \to (q_1, 1, 1z_0) \to (q_1, \varepsilon, z_0) \to (q_2, \varepsilon, z_0) \to V$

$q_0, \varepsilon, 1111z_0$ $\quad (q_1, 1, 111z_0) \to (q_1, \varepsilon, 11z_0) \to X$

$q_1, \varepsilon, 1111z_0$

X

chiamiamo **computazione** una sequenza di mosse, non necessariamente di successo. Si hanno alcune proprietà:

- se una se una sequenza di ID è lecita per un PDA P allora è lecita anche la sequenza di Id ottenuta concatenando $w \in \Sigma^*$ in ogni ID
- se una se una sequenza di ID è lecita per un PDA P e resta una coda di input non consumata allora posso rimuovere tale coda in ogni ID e ottenere un'altra sequenza lecita

- se una se una sequenza di ID è lecita per un PDA P allora è lecita la sequenza ottenuta aggiungendo $\gamma \in \Gamma^*$ in coda alla terza sequenza di ogni ID

del resto però:

$$(q, Xw, \alpha\gamma) \overset{*}{\underset{P}{\vdash}} (p, Yw, \beta\gamma) \not\rightarrow (q, X, \alpha) \overset{*}{\vdash} (p, y, \beta), \;\; x, w, y \in \Sigma^* \; \alpha, \beta, \gamma \in \Gamma^*$$

per queste proprietà valgono i seguenti teoremi:

Teorema 8. *per la seconda: Se $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ è un PDA e $(q, Xw, \alpha) \overset{*}{\underset{P}{\vdash}} (p, Yw, \beta)$ allora vale anche:*

$$(q, X, \alpha) \overset{*}{\underset{P}{\vdash}} (p, Y, \beta)$$

Teorema 9. *per la prima e la terza: Se $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ è un PDA e $(q, X, \alpha) \overset{*}{\underset{P}{\vdash}} (p, Y, \beta)$ allora:*

$$\forall \gamma \in \Gamma^* \;\; vale \;\; anche \;\; (q, Xw, \alpha\gamma) \overset{*}{\underset{P}{\vdash}} (p, Yw, \beta\gamma)$$

Si definiscono due modalità di accettazione per i PDA:

1. **per stato finale:** sia $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ si ha che:

$$L(P) = \{w \in \Sigma^* | \, (q_0, w, z_0) \overset{*}{\underset{P}{\vdash}} (q, \varepsilon, \alpha)\}$$

   con $q \in F$ e $\forall \alpha \in \Gamma^*$

2. **per stack vuoto:** sia $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ si ha che:

$$N(P) = \{w \in \Sigma^* | \, (q_0, w, z_0) \overset{*}{\underset{P}{\vdash}} (q, \varepsilon, \varepsilon)\}$$

   con $q \in Q$ e in questo caso l'insieme degli stati finali F non ha alcuna influenza

In realtà si ha che la classe di linguaggi accettati dai PDA per stato finale è uguale a quella per stack vuoto, anche se passare da un tipo all'altro di PDA è complesso. SI ha il seguente teorema per la trasformazione:

Teorema 10. *se* $L = N(P_N)$ *per un PDA* $P_N = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ *allora* $\exists$ *PDA* P_F *tale che* $L = L(P_F)$

Dimostrazione. Sia $x_0 \in \Gamma$, che indica la fine dello stack di P_F. Si ha:

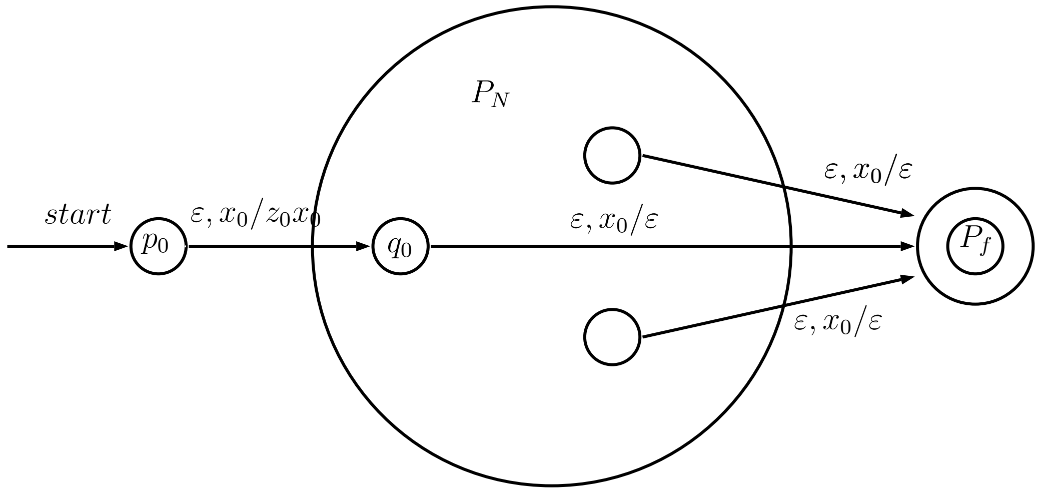
$$\delta(p_0, \varepsilon, x_0) = \{(q_0, z_0x_0\}$$

e:

$\forall q \in Q$, $\forall a \in \Sigma \cup \{\varepsilon\}, , \forall y \in \Sigma :$ $\delta_F(q, a, y)$ contiene tutte le coppie di $\delta_N(q, a, y)$

$$\forall q \in Q, \delta_F(q, \varepsilon, x_0) = \{(P_F, \varepsilon)\}$$

quindi graficamente:



Bisogna dimostrare che effettivamente $w \in L(P_F) \longleftrightarrow \in N(P_N)$.

se $w \in N(P_N)$ $\exists$ una sequenza di ID $(q_0, w, z_0) \overset{\overset{*}{\vdash}}{P_N} (q, \varepsilon, \varepsilon)$ per un qualche $q \in Q$:

$$(q_0, w, z_0x_0) \overset{\overset{*}{\vdash}}{P_N} (q, \varepsilon, x_0)$$

inoltre:

$$(q_0, w, z_0x_0) \overset{\overset{*}{\vdash}}{P_F} (q, \varepsilon, x_0)$$

e quindi:

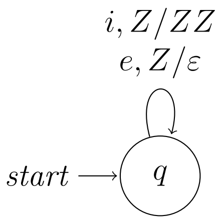
$$(p_0, w, x_0) \overset{\vdash}{P_F} (q_0, w, z_0x_0) \overset{\overset{*}{\vdash}}{P_F} (q, \varepsilon, x_0) \overset{\vdash}{P_F} (P_f, \varepsilon, \varepsilon)$$

solo se togliendo il primo e l'ultimo passo di P_F ripercorro all'indietro quanto scritto sopra. $\square$

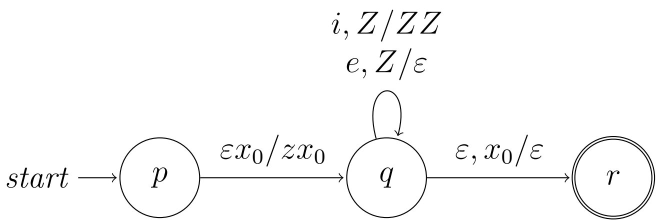
Esempio 90. *trasformazione da accettante per stack vuoto a accettante per stato finale. Siano:*

$$\Sigma = \{i, e\}$$
$$P_n = (\{q\}, \{i, e\}, \{Z\}, \delta_M, q, Z)$$
$$\delta_N(q, i, Z) = \{(q, ZZ)\}$$
$$\delta_N(q, e, Z) = \{(q, \varepsilon)\}$$

quindi:



quindi inseriamo una Z quando leggiamo i e ne rimuoviamo una se leggiamo e e si parte con una Z nello stack.
Costruisco ora il PDA P_F che accetta lo stesso linguaggio ma per stato finale, introduco lo stato iniziale p e quello accettante r, uso x_0 come segnale della fine dello stack:



e si ha formalmente:

$$P_F = (\{p, q, r\}, \{i, e\}, \{Z, x_0\}, \delta_F, p, x_0, \{r\})$$

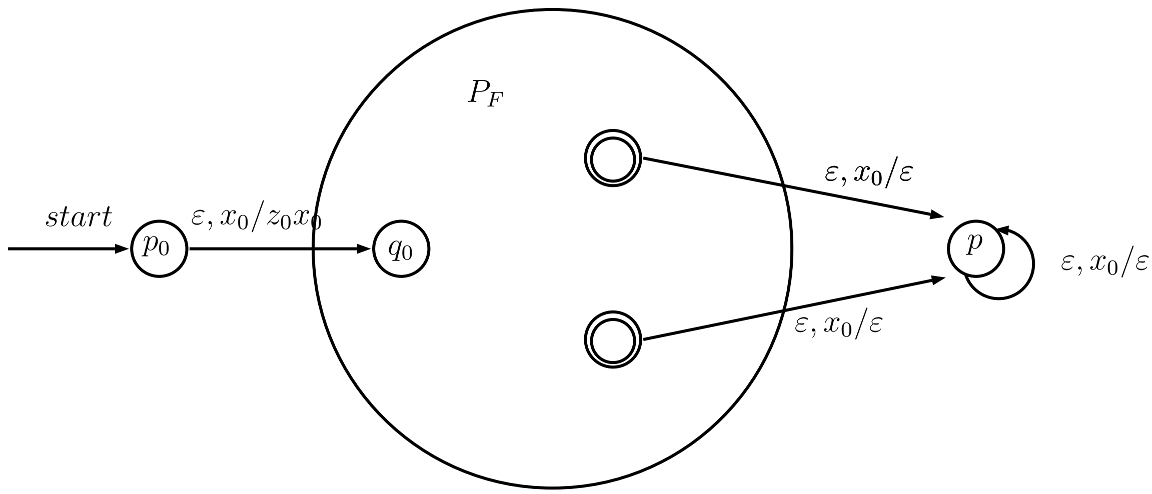
con δ_F che rappresenta le seguenti quattro regole:

1. $\delta_F(p, \varepsilon, x_0) = \{(q, Zx_0)\}$ *regola che fa partire P_F con x_o come segnalatore dello stack*
2. $\delta_F(p, i, Z) = \{(q, ZZ)\}$ *regola che inserisce Z quando si ha i simulando P_N*
3. $\delta_F(p, e, Z) = \{(q, Z\varepsilon)\}$ *regola che rimuove Z quando si ha e simulando P_N*

4. *$\delta_F(p, e, x_0) = \{(r, \varepsilon)\}$ regola che permette a P_F di accettare quando P_N esaurisce lo stack*

Si può anche effettuare la trasformazione inversa:

Teorema 11. *Sia $P_F = (Q, \Sigma, \Gamma, \delta_f, q_0, Z_0, F)$.
Si aggiunge una transizione ε a un nuovo stato p da ogni accettante di P_F. quando si ha p P_N svuota lo stack senza consumare input. Quindi se $P : F$ entra in uno stato accettante dopo aver consumato l'input w, P_N svuota lo stack dopo aver consumato w. Per evitare che si svuoti lo stack per una stringa non accettata uso x_0 per indicare il fondo dello stack. Il nuovo P_N parte da p_0 che ha il solo scopo di inserire il simbolo iniziale di P_F e passare al suo stato iniziale. Si ottiene quindi:*



e si ha formalmente:

$$P_F = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{x_0\}, \delta_N, p_0, x_0)$$

dove δ_N è così definita:

1. *$\delta_N(p_0, \varepsilon, x_0) = \{(q_0, Z_0x_0)\}$ inserisce il simbolo iniziale di P_F nello stack e va allo stato iniziale di P_F*

2. *$\forall q \in Q$ ogni simbolo di input $a \in \Sigma$, compreso l'input vuoto, e $\forall y \in \Gamma$, $\delta_N(q, a, y)$ contiene tutte le coppie di $\delta_F(q, a, y)$. Quindi P_N simula P_F*

3. *per tutti gli stati accettanti $q \in F$ e i simboli di stack $y \in \Gamma$, compreso x_0, si ha che $\delta_N(q, \varepsilon, y)$ contiene (p, ε), quindi ogni volta che P_F accetta P_N inizia scaricare lo stack senza consumare ulteriori input*

4. per tutti i simboli di stack $y \in \Gamma$, compreso x_0, si ha che $\delta_N(q, \varepsilon, y) = \{(p, \varepsilon)\}$, quindi giunti allo stato p, ovvero quando P_F ha accettato, P_N elimina ogni simbolo nel suo stack fino a svuotarlo

inoltre formalmente voglio dimostrare che:

$$w \in L(P_F) \to w \in N(P_N)$$

e quindi ho le seguenti mosse:

$$(q_0, w, z_0) \overset{\overset{*}{\vdash}}{P_F} (q, \varepsilon, \alpha) \; q \in F, \;\; \alpha \in \Gamma^*$$

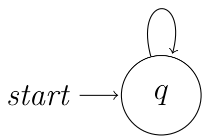
$$(p_o, w, x_0) \vdash (q_0, w, z_0x_0) \overset{\overset{*}{\vdash}}{P_N} (q, \varepsilon, \alpha, x_0) \overset{\overset{*}{\vdash}}{P_N} (p, \varepsilon, \varepsilon)$$

Esempio 91. *si ha una CFG $G = (\{i, e\}, \{a, b, 0, 1, *, +, (,)\}, P, E)$ con:*

$$P : I \to a|b|Ia|Ib|I0|I1$$

$$E \to E + E|E * E|(E)$$

si ha il PDA $P_G = (\{q\}, \Sigma, \Sigma \cup \{i, e\}, \delta, q, E)$:



si ha quindi:

$$\delta(q, \varepsilon, i) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$$

$$\delta(q, \varepsilon, E) = \{(q, I), (q, E + E), (q, E * E), (q, I(E))\}$$

$$\delta(q, a, a) = \{(q, \varepsilon)\}$$

$$\delta(q, b, b) = \{(q, \varepsilon)\}$$

$$\cdots = \{(q, \varepsilon)\}$$

quindi si ha:

$$E \to E + E \to i + E \to a + (E) \to a + (i) \to a + (i0) \to a + (b0)$$

$$(q, a+(b0), E) \vdash (q, a+(b0), E+E) \vdash (q, a+(b0), i+E) \vdash (q, a+(b0), a+E)$$
$$\vdash (q, +(b0), +E) \vdash (q, (b0), E) \vdash (q, (b0), (E)) \vdash (q, b0), E))$$
$$\vdash (q, b0), i)) \vdash (q, b0), i0)) \vdash (q, b0), b0) \vdash (q, 0), 0))$$
$$\vdash (q,),)) \vdash (q, \varepsilon, \varepsilon)$$

questo esempio è generalizzabile ad ogni CFG:

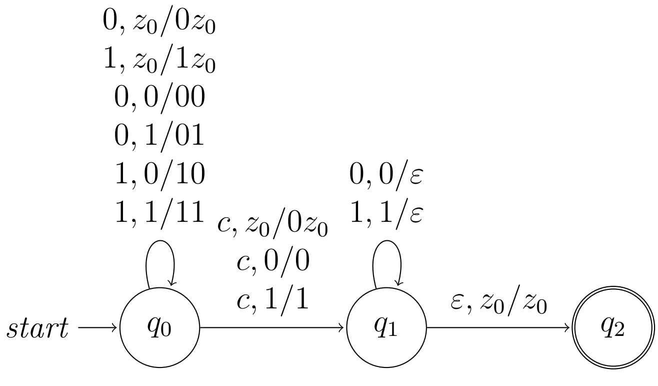
$$\exists\ PDA\ Q = (\{q\}, T, V \cup T, \delta, q, S) \text{ tale che } N(Q) = L(G)$$
$$\forall A \in V\ \ \delta(q, \varepsilon, A) = \{(q, \beta) |\ A \to B \text{ e' una produzione di G}\}$$
$$\forall a \in T\ \ \delta(q, a, a) = \{(a, \varepsilon)\}$$

Teorema 12. *sia $G = (V, T, P, S)$ una CFG:*

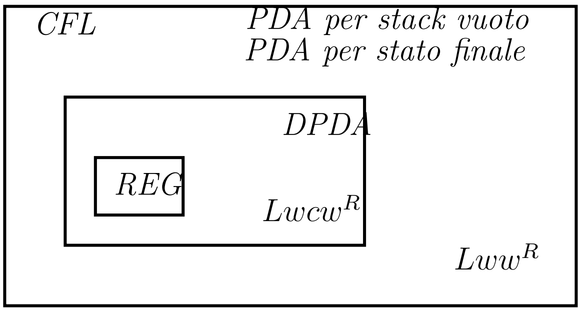
Questo dimostra che ogni CFL può essere accettato da un PDA accettante per stack vuoto. Per il teorema visto in precedenza, posso sempre costruire un altro PDA accettante per stati finale. I PDA accettano tutti e soli i linguaggi CF. Mostrare che accettano solo linguaggi di tipo 2 è complicato. Un tipo di PDA interessante, soprattutto per i parse, è il PDA deterministico, il **DPDA**.
Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ è deterministico se:

1. $|\delta(q, a, x)| \leq 1\ \forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall x \in \Gamma$
2. se $|\delta(q, a, x)| \neq 0$ per qualche $a \in \Sigma$ allora $|\delta(q, \varepsilon, x)| = 0$

Esempio 92. *abbiamo il linguaggio $L_{wcw^R} = \{wcw^R |\ w \in \{0, 1\}^*\}$*
Gli automi a pila deterministici non riconosco tutti i CFL, ma solo una classe strettamente più piccola. Ad esempio non potrebbero riconoscere il linguaggio delle palindrome senza "il segnalibro" c. SI ha quindi:



quindi:



si ha infatti il seguente teorema:

Teorema 13. *$L \in REG \rightarrow \exists PDA\ P$ tale che $L = L(P)$*

Dimostrazione.

$$L \in REG \rightarrow \exists DFA\ A = (Q, \Sigma, \delta_A, q_0, F) \text{ tale che } L = L(A)$$

costruisco il DPDA $P = (Q, \Sigma, \{z_0\}, \delta_p, q_0, z_0, F)$ con:

$$\delta_p(q, a, z_0) = \{p, z_0\}\ \forall p, q \in Q \text{ tali che } \delta_A(q, a) = 0$$

vale:

$$(q_0, w, z_0) \overset{A}{\overset{\vdash}{P}} (p, \varepsilon, z_0) \longleftrightarrow \overset{\wedge}{\delta}_A (q_0, w) = p$$

□

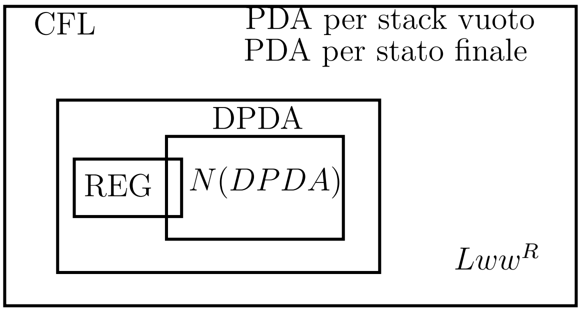
si ha inoltre il seguente teorema:

Teorema 14. *L è $N(P)$ per un DPDA P sse L è $L(P^{'})$ per un DPDA $P^{'}$ e L ha le proprietà di prefisso* ***prefix-free***

definiamo così la proprietà di prefisso:

$$\nexists x, y \in L \text{ tali che } x \neq y \text{ e } x \text{ è prefisso di } y$$

per esempio $L = \{0\}^0 = \{\varepsilon, 0, 00, 000, ...\}$ non ha la proprietà di prefisso. Osserviamo che se la stringa vuota appartiene al linguaggio, tale stringa è prefissa di tutte le altre e quindi il linguaggio non può avere la proprietà di prefisso. Affermiamo che L è regolare, quindi è accettato da un DPDA per stati finali ma non da uno per stack vuoto. Completiamo il diagramma precedente sulle classi di linguaggi:



SI ha che L_{wcw^R} gode della proprietà di prefisso:

$$y = wcw^R \in L \ Se \ x \neq y, \text{ prefisso di } y, x \notin L$$

tornando alle grammatiche si hanno ora due teoremi:

Teorema 15. *se $L = N(P)$ per un DPDA P, allora L ha una CFG non ambigua*

Teorema 16. *se $L = L(P)$ per un DPDA P, allora L ha una CFG non ambigua*

dimostriamo il secondo:

Dimostrazione. $L = L(P)$ per un DPDA P, costruiamo $L' = L$, quindi L' ha la proprietà di prefisso. Esiste quindi un DPDA P' tale che $L'N(P)$, esiste quindi per il teorema sopra una CFG G' tale che $L(G') = L'$ che non è ambigua.
Costruiamo G per L con le stesse produzioni di G' più $\$ \to \varepsilon$, applicata solo all'ultimo passo. □

Vogliamo scoprire se è vero il viceversa: per ogni L che ha una CFG non ambigua è vero che L è accettato da un DPDA? No, mostriamo infatti un controesempio:
$S \to 0S0|1S1|\varepsilon$ produce L_{ww^R} che non è accettato da alcun PDA